University of Ljubljana
Faculty of Computer and Information Science

Kristijan Shirgoski

Computer-vision based polyp detection in underwater images

bachelor's thesis

undergraduate university study programme
computer and information science

Mentor: doc. dr. Matej Kristan

Ljubljana 2015

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Kristijan Shirgoski

Zaznavanje polipov z računalniškim vidom v podvodnih slikah

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Matej Kristan

Ljubljana 2015

The Faculty of Computer and Information Science issues the following thesis:

Address the problem of polyp detection in underwater images. Based on the overview of the recent state-of-the-art in computer-vision-based object detection and on the specific visual properties of polyp images, propose the most appropriate methodology to tackle the problem of polyp detection. The developed approach has to account for the fact that the visual properties of polyps may vary due to uncontrolled environment in which the images are taken and that the images are typically densely populated by the polyps. Construct and annotate a dataset for realistic underwater images of polyps and use it to analyze the advantages and drawbacks of the proposed polyp detector.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

V nalogi naslovite problem detekcije polipov v podvodnih slikah. Preglejte sorodna dela na področju detekcije objektov z računalniškim vidom in na podlagi specifičnih lastnosti slik polipov izberite najprimernejši pristop k detekciji. Upoštevajte, da so lahko na slikah polipi zelo na gosto posejani, ter da se njihov izgled zaradi nekontroliranih pogojev zajema slik lahko spreminja. Za namen analize algoritma zgradite in anotirajte zbirko tipičnih slik polipov. S pomočjo anotirane zbirke analizirajte glavne dele vašega algoritma ter izpostavite prednosti ter slabosti predlagane metode.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Kristijan Shirgoski sem avtor diplomskega dela z naslovom: *Zaznavanje polipov z računalniškim vidom v podvodnih slikah*

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Mateja Kristana,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 1. septembra 2015 Podpis avtorja:

First and foremost, I would like to thank my mentor doc. dr. Matej Kristan for his tremendous effort and brilliant ideas during the realization of this thesis. I truly enjoyed researching under his supervision.

Furthermore, I gratefully acknowledge Rok Mandeljc and Domen Tabernik for their constructive help and time on the thesis.

In addition, I would like to show my appreciation to my peers Peter, Vanja, Žiga, Matic, Jan, Kristijan, Vase and my best friend Igor for their invaluable and honest friendship and support throughout the studies.

Finally, I wish to express my deepest gratitude to my parents and sister for their unconditional love and endless encouragement and support. I owe everything to them.

Посветено на мојата фамилија.

Contents

Table of acronyms

acronym	english	slovensko
ML	machine learning	strojno učenje
ACF	aggregated channel features	agregirane značilnice po kanalih
CNN	convolutional neural network	konvolucijska nevronska mreža
SVM	support vector machine	metoda podpornih vektorjev
HOG	histogram of oriented gradients	histogram orientiranih gradientov
MATLAB	MATrix LABoratory	MATLAB
BP	back-propagation	vzvratno razširjanje napake
FP	forward-propagation	razširjanje napake naprej

Povzetek

Detekcija objektov je priljubljena tematika na področjih računalniškega vida in strojnega učenja. Reševanje problemov detekcije objektov predstavlja precejšenj izziv in v literaturi obstaja veliko različnih pristopov. Pristopi se konceptualno precej razlikujejo, različne pa so tudi njihove časovne zahtevnosti. Cilj diplomske naloge je ustvariti splošen pristop za zaznavanje objektov v sliki, v našem primeru polipe klobučnjaku *Aurelia aurita*. Za te polipe je značilno, da se na gosto širijo preko ostrige. Ena od značilnih tematik pri detekciji objektov je iskanje določenih regij v sliki, ki nas zanimajo. Ponavadi iščemo regije različnih velikosti. Zato smo predlagali uporabo modela agregirane značilnice po kanalih (ACF), ki se je učil na anotaciji iz naše zbirke. Iz vsake regije moramo izluščiti podatke, na podlagi katerih posamezni regiji določimo lastnosti ali karakteristike. V tej diplomski nalogi smo to izvedli s pomočjo konvolucijske nevronske mreže (CNN), ki je bila trenirana na podatkovni zbirki MNIST. Poleg tega sta za klasifikacijo in ocenjevanje njene pravilnosti uporabljena binarni klasifikator podpornih vektorjev (SVM) z linearnim jedrom ter $L2$ regularizirana logistična regresija. Zelo verjetno je, da vsaki anotaciji pripada več regij, ki jih ACF predlaga. Zato je potrebno veliko teh regij odstraniti s pomočjo metode tlačenja nemaksimumov, ki se naivno osredotoča na tiste regije, ki imajo višje ocene. Algoritme, ki smo jih uporabljali, so bili učeni in preizkušeni na novi zbirki podatkov, ki je sestavljena iz skoraj 40000 pravokotnih anotacij v 35 slik. Dosegli smo zelo obetavne rezultate ter analizirali prednosti in slabosti našega pristopa.

Ključne besede: detekcija objektov, računalniški vid, strojno učenje, ACF, konvolucijska nevronska mreža, metoda podpornih vektorjev, tlačenje nemaksimumov.

Abstract

Object detection is a popular topic in computer vision and machine learning. Numerous approaches have been proposed in literature to address the challenging task of general object detection. The approaches vary conceptually as well as in the level of computational intensity. The goal of this thesis was to develop a pipeline of state-of-the-art algorithms to detect polyps of the *Aurelia aurita* jellyfish, which are densely spread across oysters. In object detection problems, a mandatory task is searching the image for regions of interest, preferably of several sizes. We propose a trained aggregated channel features (ACF) model to do that. In order to later classify these regions, first they need to have some features or characteristics extracted from them. In this thesis, this is performed by a convolutional neural network (CNN) trained on the MNIST dataset. Furthermore, a binary support vector machines (SVM) classifier with linear kernel and $L2$-regularized logistic regression is used to classify the features and determine the probability of correctly classifying them. It is very likely that several regions are proposed for each ground truth, so the regions must undergo a non-maximum suppression which uses the probability outputs from the logistic regression to group the local regions together, greedily prioritizing based on the probability distribution. The algorithms were trained and tested on 35 images consistent of nearly 40000 rectangle annotations from a newly annotated dataset. We have achieved very promising results and analyzed the strengths and weaknesses of our approach.

Keywords: object detection, computer vision, machine learning, ACF, CNN, SVM, regression, non-maximum suppression.

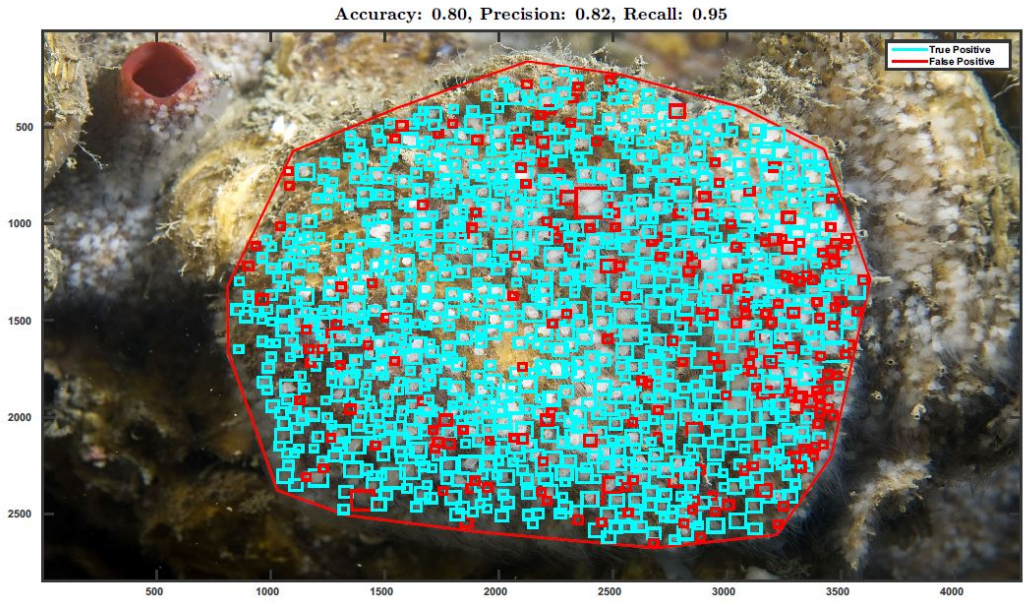
Razširjeni povzetek

Detekcija objektov je proces identifikacije posameznih objektov v slikah oziroma v video posnetkih. Je zelo široko raziskovano področje, za katerega obstaja ogromno število različnih pristopov. Njihova praktična vrednost je velika, uporablja pa jih velik delež aplikacij, ki so že vgrajene v naše mobilne telefone, varnostne kamere, avtomobile, robote in podobno. Kljub temu, da je to raziskovalno področje doseglo že izjemno kvalitetne rezultate in so nekateri izmed njih celo boljši od rezultatov človeških zaznav, je možnosti za izboljšave in nove raziskave ogromno. Področje je zelo povezano in odvisno od strojne opreme in kvalitete informacij, ki jih dobimo iz okolja. Napredki na teh področjih še bolj doprinesejo k razvoju in kakovosti sodobnih rešitev za detekcijo objektov. Sodobne aplikacije zelo zaostajajo za tem, kar človek lahko doseže z svojim možgani v domeni zaznavanja objektov. Ponavadi so te aplikacije prilagojene zgolj na iskanje nekaterih zelo specifičnih objektov na sliki. Rešitve, ki bi bile zmožne sprejeti kakršno koli sliko in v realnem času natančno prepoznati kakšne objekte le-ta vsebuje, še vedno ne obstajajo. Vendar, ljudje niso vsemogočni. Velika pomanjkljivost človeka je nemožnost zaznavanja in preštevanja objektov, ki se na slikah zelo velikokrat pojavijo. Zato je velik izziv in motivacija razviti dodatke oziroma aplikacije, ki naredijo to nalogo namesto ljudi. V aplikacijah, ki zaznavajo objekte, je ponavadi prvi problem najti regije, ki so potencialni objekti. Včasih so se ljudje lotevali reševanja teh problemov z metodo drsnega okna. Ta metoda požrešno preizkuje regije različnih velikosti in jih klasificira. V sodobnih rešitvah se tega izognemo s pomočjo številnih dodatnih postopkov, kot je na primer upoštevanje informacije o robovih [54], segmentacija [47, 48] in podobno. Zelo pomemben del teh rešitev je pridobivanje značilnic iz teh

regij. Pri detekciji objektov imajo ljudje zelo veliko prednost pred aplikacijami, saj imajo veliko več predznanj in informacije pridobivajo na različne načine. S slike je na primer zelo težko dobiti informacijo o tem, koliko so piksli medsebojno oddaljeni, razen če je okolje močno omejeno ali imamo njihov 3-dimenzionalni model. Sodobne rešitve ponavadi računajo številčne predstavitve, kot so histogrami HOG [14] ali druge značilnice kot so SIFT [34], SURF [3] ali nevronske mreže [30]. Te značilnice so potem klasificirane s pomočjo veliko različnih algoritmov, kot so klasifikacijska drevesa [40], metoda podpornih vektorjev (SVM) [12] in tako naprej.

Cilj diplomske naloge je ustvariti splošen pristop za zaznavanje objektov v sliki. Da bi to dosegli, smo izbrali podmnožico navedenih algoritmov. Odločili smo se, da bomo poskusili dobiti regije, na katerih so potencialni objekti, s pomočjo agregirane značilnice po kanalih (ACF) [16]. Model smo trenirali na približno 1300 pozitivnih in 2150 negativnih primerih. Merili smo kvaliteto regij z računanjem ulomkov preseka in unije ploščine teh regij. Po podrobni vizualni analizi smo se odločili, da je pravi prag 0.2. Vse regije, ki so imele prag večji od 0.2, so bile označene kot pozitivne, vse ostale kot negativne.
Naslednji korak je bil izluščevanje karakteristik oziroma značilnosti teh regij. Uporabili smo metodo konvolucijskih nevronskih mrež (CNN), ki je široko uporabljena metoda za pridobivanje značilnic. Mreža je bila trenirana na podatkovni zbirki MNIST in vsebuje 7 plasti, od katerih sta 2 konvolucijski, 2 podvzorčni, 1 ReLU in 2 v celoti povezani plasti. Izhod iz mreže so $N \times 10$ vektorji, kjer je N število regij. Te značilnice smo potem klasificirali z metodo podpornih vektorjev (SVM), bolj natančno z linearnim jedrom z $L2$ logistično regresijo, ki je računala verjetnosti, da so posamezne klasifikacije natančne. Odvečne regije so potem filtrirane z uporabo metode tlačenja nemaksimumov z pragom 0.1. Naš pristop je bil učenj in testiran na novi zbirki podatkov, ki smo jo sami anotirali. V njej je 35 slik, ki vsebujejo skoraj 40000 pravokotnih anotacij. Slike so zajete pod vodo in polipi so razpršeni po ostrigah. Anotacije so bile zgolj na polipih, ki se nahajajo na ostrigah, ker so bili ostali deli slike preveč zamegljeni in tudi za človeka dvoumni. Zato smo testiranje omejili samo na ostrige, ki so bile anotirane kot poligoni. Analizirali smo glavne značilnosti zbirke, kot je na primer velikost anotacij, ker so nas zanimale

velikosti polipov. Analizirali smo tudi kako so polipi v povprečju razpršeni po sliki. Nato smo analizirali še rezultate posameznih algoritmov našega pristopa. Ugotovili smo, da ACF povprečno najde 96.4% regij v sliki, ki imajo vsaj 0.2 ujemanje z anotacijami. Njihovo povprečno ujemanje je 0.54, kar je zelo dobro. Naredili smo veliko različnih eksperimentov. Poskusili smo centralizirati podatke in povečevali smo regije za 5%, 10% in 15%, da bi videl, če dodatne informacije o okolju pomagajo nevronskim mrežam in klasifikatorju. Vsak eksperiment smo ponovili tudi z ℓ^2 normalizacijo. Ugotovili smo, da povečevanje regije za 15% da najboljše rezultate. S 95 odstotnim zaupanjem ta model doseže klasifikacijsko točnost $\mathbf{0.71 \pm 0.12}$, natančnost $\mathbf{0.65 \pm 0.24}$ in priklic $\mathbf{0.89 \pm 0.12}$. V povprečju se dejansko število polipov razlikuje za **194(16%)** od števila detekcij. Slika 1 vizualno prikazuje kvaliteto detektorja na eni izmed manj zameglenih slik. Analizirali smo tudi, če je v modelu kakšna pristranskost in izkazalo se je, da gre za dinamično pristranskost. To pomeni, da je možno ta model še izboljšati. Naš pristop smo primerjali z ACF detektorjem in naš model je uspel doseči občutno boljše rezultate.



Slika 1: Prikaz vseh detekcije v eni sliki. Modre pravokotnike so pravilno klasificirane polipe, rdeče pa regije ki so napačno klasificirane kot polipe.

Chapter 1

Introduction

Object detection is a very prominent and challenging problem. It is well known in computer vision that the solutions to these problems are very specific and have to be carefully tailored and adapted according to the problem. One cannot simply plug in any kind of image to an object detection solution and hope to have localized and classified all objects accurately. Most frequently these algorithms expect a fairly similar and specific kinds of images as input. In this thesis, the images are taken underwater and the objects that are to be detected are extremely dense and have diverse forms, making the problem even more challenging and appealing. Therefore, the motivation of achieving a robust solution for this problem is huge.

The main goal of this thesis was was addressing the problem of detection of polyps by computer vision methods. We propose a pipeline of several potent computer vision methods which have recently had a major influence in the field of computer vision. Such pipeline can be applicable to an immense variety of problems related to counting masses of objects, since these problems are too exhausting for human operators. Humans tend to make errors in demanding counting problems. An example is biology and medicine, where automated object detection and counting can be used to count images crowded with cells, bacteria and objects similar to them, or in astronomy to count stars and other extraterrestrial objects.

1.1 Related Work

The research done in the wide domain of object detection has produced a lot of practically useful approaches and applications to support humans to another extent. The break-through applications were using the sliding window method [49]. This standard technique for object detection scans the image with windows of different scales and aspects which are then further processed and classified. Another example of an application using this method is Aggregated Channel Features (ACF) [16] that has been used only as a pedestrian detector [16] and a similar approach is applied for face detection [53]. However, object detection solutions that use this technique have a huge drawback of having to classify a tremendous amount of regions. Consequently, a lot of work has been done lately to avoid this detriment. Recent solutions output only a small subset of those regions by obtaining them using edge information [54] together with edge densty and superpixels straddling [1], segmentation [47, 48] and so on. The research that is mostly related to region proposals is [1], because their main intention was to quantify how likely it is for an image window to contain an object of any class.

Another important aspect of object detection is extracting features (or characteristics) for these regions in order to later classify them. Some powerful methods for extracting features are scale-invariant feature transform (SIFT) [34], speeded up robust features (SURF) [3], histogram of oriented gradients (HOG) [14] and local intensity order pattern (LIOP) [50]. Furthermore, inspired by the recent success of the research [32, 31, 11, 36] in convolutional meural networks (CNN), mostly by [30] that sparked a lot of research because of their impressive results, [8] for their GPU implementation of CNNs and [7] that also confirmed that CNNs are indeed powerful feature extractors, these networks have been immensely used for feature extraction. A lot of constructive guidelines have been used from those sources. In addition, statistical procedures such as principal component analysis (PCA) [29] or linear discriminant analysis (LDA) [44] are sometimes applied to the features to reduce their dimensionality.

Another very important topic in object detection and machine learning is classification. Observing informative features in regions is useless without a method

that can determine the class that the features belong to. A small subset of these algorithms include decision trees [40], support vector machines (SVM) [12], Naive Bayes [42], k-nearest neighbors (KNN) [2] and others.

Our approach is fairly similar to [22], with a major difference in the region proposals method. They use a selective search to obtain region proposals, train a fine-tuned CNN and properly adapt it to the new task and optimize a SVM classifier for each class. As a whole, this thesis' problem is the most similar with [41, 33, 5] because their work includes detecting and counting objects in dense environments. Their approaches are however different than ours. The goal of [5] is tracking people and counting them by segmenting the crowd into components of homogeneous motion using a dynamic textures motion model, then they extract holistic features for each segment and use them in a Gaussian Process regression to learn the correspondence between features and the number of people. In addition, [33] independently do the same task and further test it on counting cells while not focusing on detecting and localizing the objects, but rather by estimating their density with a regularized risk quadratic cost function. Also, [41] try to estimate the density of a moving crowd of people by using a joint energy function combining crowd density estimation and the localization of individual people.
Each of the many methods used in object detection has its own pros and cons, making the design of object detection solutions very diverse and interesting.

1.2 Contributions

The first contribution of this thesis is proposing a new approach for detecting densely distributed objects in a cluttered and visually ambiguous environment. The first part of the pipeline is fairly unique, since Aggregated Channel Features (ACF) [16] is not commonly used for region proposals and had been mostly used for pedestrian detection by far. We have successfully trained an ACF model and integrated it into the pipeline to propose regions that might contain the desired polyps. The second and third part of the pipeline, the Convolutional Neural Networks (CNNs) [30] and the Support Vector Machines (SVM) [12] had faced

new fairly blurry water frontiers, giving these results additional significance.

The second contribution of this thesis is a newly annotated dataset made up of 39 images of oysters covered with a huge amounts of white *Aurelia aurita* polyps [25]. We would like to acknowledge [25, 26, 35] for recording the images of the dataset. The dataset consists of 44310 carefully annotated rectangles, which are polyps lying on the oysters. This dataset with such a great detail of annotation is the first of its kind.

Finally, one of the perks of this approach is the broad spectrum of object detection problems it can be applied to, making it practically useful. The same approach can be used to detect various densely distributed objects in order to measure their density and ease the distribution monitoring and further statistical analysis.

1.3 Structure

The remainder of the thesis is structured as follows. Chapter 2 explains in detail the theory behind the methods used. It consists of an overview of the ACF [16] algorithm that outputs regions of interest, description of the CNN [24] features extraction method which calculates characteristics of the proposed ACF regions and lastly an thorough explanation of SVM [27] along with its perks and detriments. Chapter 3 describes the pipeline constructed of the aforementioned techniques and how they are connected. Our approach is experimentally validated in Chapter 4 and discussion of the results is provided, along with details about the dataset, implementation and parameters. Chapter 5 is the final chapter of the thesis that concludes the research and presents ideas for future work.

Chapter 2

Methods used

2.1 Aggregated channel features

Aggregated Channel Features (ACF) [16] is a fairly new detector introduced in 2014. Its intended use was as a pedestrian detector. Being simple and conceptually straightforward, it consists of several phases as shown in Figure 2.1. First several channels C are computed from the input image I, i.e., $C = \Omega(I)$, where $\Omega()$ is the channel function. Furthermore every block of pixels in the channels C are summed and smoothed, resulting in smaller resolution channels. Those smaller channels are then vectorized and the features are single pixel lookups in the aggregated channels. Finally, these features are used in a boosting process in order to train decision trees [40]. Using the built model these regions are either outputted or dismissed. The detector works with a multiscale sliding window technique.

Figure 2.1: Visualization of the processes behind the simple yet effective ACF detector. Image taken from [16].

The ACF uses three types of channels: normalized gradient magnitude [14], histogram of oriented gradients (HOG) [14] and LUV [20] color channels.
The normalized gradient magnitude channel yields a great amount of information, because it contains the gradient strengths of the image. Gradients are very important information in an image because the gradient points in the direction of greatest intensity change, as shown in Figure 2.2.

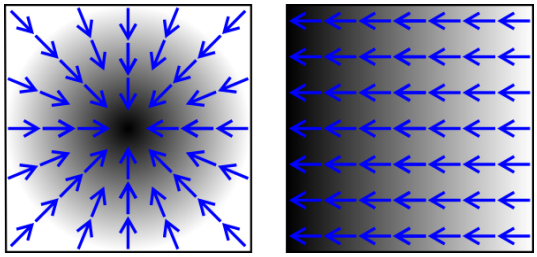


Figure 2.2: Intuitive representation of the gradients represented by blue arrows that correspond to change of intensity. Image taken from [52].

The gradient strength is therefore defined by its magnitude

$$|| \bigtriangledown f|| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}. \tag{2.1}$$

The gradient direction is

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial x} / \frac{\partial f}{\partial y}\right). \tag{2.2}$$

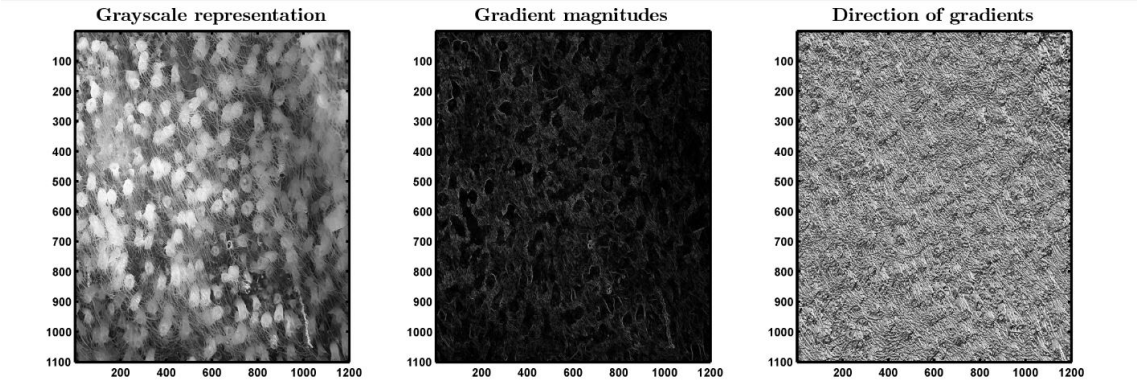


Figure 2.3: Equations (2.1) and (2.2) applied to a part of an image from the new dataset.

After obtaining the gradient magnitudes M by using (2.1), normalized gradient magnitudes $\widetilde{M}$ are computed by

$$\widetilde{M}(i,j) = \frac{M(i,j)}{\overline{M}(i,j) + 0.005}, \tag{2.3}$$

where $\overline{M}$ is the average gradient magnitude in each image patch of size 11×11 and is computed by convolving M with an $L1$ normalized 11×11 triangle filter. The second popular feature descriptor in ACF is the histogram of oriented gradients (HOG) [14]. There are n HOG channels set in the detector, which means that there are n different directions for which the gradient is computed. It is computed on a dense grid of uniformly spaced cells and uses overlapping local contrast normalization for improved accuracy.

The first step of computing a HOG is pre-processing the image to normalize the colors and gamma values. Dalal and Triggs point out in their research [14] that it is essential to perform a strong local normalization. They found out that normalizing each edge or cell several times with respect to different local supports boosts the results. Also, they argue that pre-processing an image before normalizing it is not mandatory because they have achieved similar results by not doing so.

The second step of the HOG computing procedure is creating and calculating cell histograms. They can be in either rectangular or radial shape. Furthermore,

weighted votes per pixel in the cells are computed based on the already calculated gradients. The histogram can be calculated on either $0-180$ or $0-360$ degrees and the votes can be cast either by using some function of the gradient magnitude or the gradient magnitude itself (which has proven to produce better results). Last but not least, the normalization step is obligatory in order to account the changes in illumination and contrast. Thus the cells are grouped into larger spatially connected blocks. The final HOG descriptor is a vector containing the components of the normalized cell histograms from all block regions. It regularly occurs that the cells overlap, thus contributing to the final descriptor more than once.
The most popular normalization method is

$$f = \frac{\mathbf{v}}{\sqrt{||\mathbf{v}||^k + e^k}}. \tag{2.4}$$

where $\mathbf{v}$ is the non-normalized vector that contains the histograms in a particular block, e is a very small constant and $||\mathbf{v}||$ is its k-norm, where k is usually 1 or 2. When $k = 2$ we call that a $L2$-norm, and when $k = 1$ it is a $L1$-norm.
Before moving to the last computed channel, it is important to mention that ACF extracts multi-scale features. To do so, it is required to compute the histograms on different scales. However, doing this would require too much computational time, which real-time detectors can not afford. That is why these histograms are approximated using half-octave pyramids [13], achieving similar results to the originals.
Last but not least, ACF uses the LUV [20] color representation as its last channel from which a feature vector is built. The uniformly distributed LUV color representation is needed because the XYZ [46] color space is based on human perception and thus it is not uniformly distributed. It takes advantage of the fact that humans tend to perceive the light within the green parts of the spectrum as brighter than red or blue light of equal strength. Therefore, Y is defined as the luminance, Z is quasi-equal to blue stimulation and X is a linear combination of cone response curves chosen to be nonnegative. The LUV color space is computed directly from

the standard XYZ color space by applying the following transformations:

$$L^* = \begin{cases} \left(\frac{29}{3}\right)^3 Y/Y_n, & Y/Y_n \leq \left(\frac{6}{29}\right)^3 \\ 116\,(Y/Y_n)^{1/3} - 16, & Y/Y_n > \left(\frac{6}{29}\right)^3 \end{cases}, \tag{2.5}$$

$$u^* = 13L^* \cdot (u' - u'_n), \tag{2.6}$$

$$v^* = 13L^* \cdot (v' - v'_n). \tag{2.7}$$

For most images $-100 \leq u^*, v^* \leq 100$ and $0 \leq L^* \leq 100$. After applying (2.8) and (2.9), the u' and v' values can be used to construct the beautiful diagram shown in Figure 2.4. The green color in the XYZ color space appears to be very dominant.

$$u' = \frac{4X}{X + 15Y + 3Z} = \frac{4x}{-2x + 12y + 3} \tag{2.8}$$

$$v' = \frac{9Y}{X + 15Y + 3Z} = \frac{9y}{-2x + 12y + 3} \tag{2.9}$$

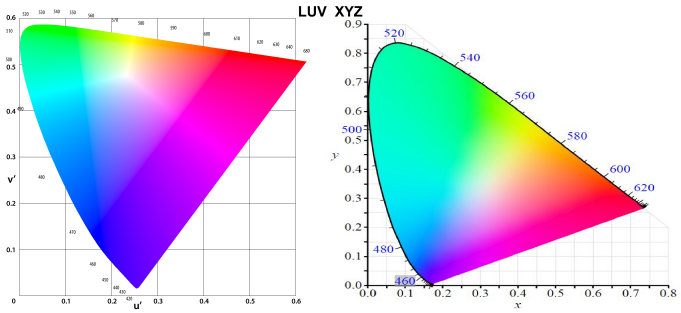


Figure 2.4: Visualization of the LUV and XYZ chromaticity diagrams. Images taken from [51].

Once the channels are computed, they are aggregated and vectorized as mentioned before. The only thing left is training an Adaboost classifier on the features constructed from these channels.
The main idea behind boosting [43] is building a classifier from a combination of

many "weak classifiers". This is called sequential learning because each iteration has its own weak classifier. One easily noticeable advantage is that boosting is very flexible since we can add many sequential weak classifiers of our choice.
Adaboost [21], short for Adaptive Boosting, is a proven invaluable machine learning algorithm, mostly known for its success in the Viola-Jones [49] face detection algorithm. Unfortunately, Adaboost is sensitive to outliers and noise in the data. As long as each subsequent weak classifier can achieve better results than random guessing, eventually the final model converges to become a strong model. Given n examples $\mathbf{x}$ and their $\mathbf{y}$ labels respectively, the Adaboost algorithm starts off by initializing the weights

$$w_{1,i} = \frac{1}{2m}, \frac{1}{2l} \quad for\ y_i = 0, 1, \tag{2.10}$$

where m is the number of negative examples and l is the number of positive examples. The following steps are then repeated T times, where t represents the current iteration:

1. Normalize the weights, so that w_t is a probability distribution,

$$w_{t,i} = \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}. \tag{2.11}$$

2. For each feature j train a classifier h_j which is restricted to using a single feature. The classification error is evaluated with respect to w_t,

$$e_t = \sum_i w_i |h_j(x_i) - y_i|. \tag{2.12}$$

3. Choose the classifier h_t with the lowest error e_t.

4. Update the weights,

$$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}, \tag{2.13}$$

where $e_i = 0$ if the example x_i is classified correctly and $e_i = 1$ otherwise and $\beta_t = \frac{e_t}{1-e_t}$. This way, the incorrect classifications get higher weights, and the correct classifications lower weights.

The final classifier is then defined as

$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & otherwise \end{cases}, \tag{2.14}$$

where $\alpha_t = \log \frac{1}{\beta_t}$. Accordingly, it is a combination of weak classifiers which are weighted according to their error.
ACF combines the Adaboost method [21] together with decision trees [40] as weak learners. When these two methods are combined, information gathered about the relative 'hardness' of each training sample at each stage of the AdaBoost algorithm is fed into the tree growing algorithm such that later trees tend to focus on harder examples. Decision trees tend to split the data into pure branches. They are easy to interpret by humans and no information is lost while building them. However, they are easily distorted to small fluctuation in the data. Boosting is used to avoid this. The idea is to build many trees so that the weighted average over all trees is insensitive to fluctuations. ACF builds up many trees (up to 2048 trees by default) and optimizes them by reweighting, scoring each trees quality and averaging over all trees using the obtained scores as weights.

2.2 Convolutional neural networks

Having described the theory behind ACF in the previous section, this section focuses on the features extractor which we use in the thesis, the convolutional neural network (CNN) [30]. This section is started with a brief overview of CNNs, followed by explanations and important concepts for each type of layer and finally completing the theory by deriving the equations for the training and test process.

2.2.1 Overview

Convolutional Neural Networks (CNN) [30] are deep feed-forward variations of multi-layer-perceptrons (MLP) [24] inspired by biological systems, more precisely by cells existant in some animals which are sensitive to receptive fields in the visual cortex and serve as excellent local filters exploiting strong correlations in the

image. Bearing a supervised learning architecture, the CNNs differ from the visual cortex by having filters which are not fixed. The weights(filters) are learned in a supervised way through back propagation. Many neurons are stacked such that they respond to overlapping regions. CNNs have another perk which is using minimal ammount of preprocessing and differ from MLPs in that way. CNNs have very sophisticated structures compared to other standard representations, consistant of many layers of non-linear feature extractors, thus making them being called deep. Recent research [9, 10] by Dan Ciresan has greatly improved the training time of CNNs by implementing them on GPU, making them even more appealing to researchers. Having a handcrafted structure, they contain a tremendous amount of parameters learned from the data. CNNs vary a lot by how the layers are realized and how they are trained. The output feature vectors intended for later classification are learned through convolution of parts of the image with filters which are then repeatedly sub-sampled and re-filtered.

Figure 2.5 illustrates a single artificial neuron. It has many inputs which are weighted by a function specified by the networks architect and the activations of the neuron are later passed to other neurons in the network. One final note before concluding this subsection is that CNNs generate strong feature vectors that are quite invariant to image distortion and position [8]. Images from their training set were being randomly translated and rotated with values sampled from a uniform distribution. The additional translations and rotations even improved their results, which is astonishing.

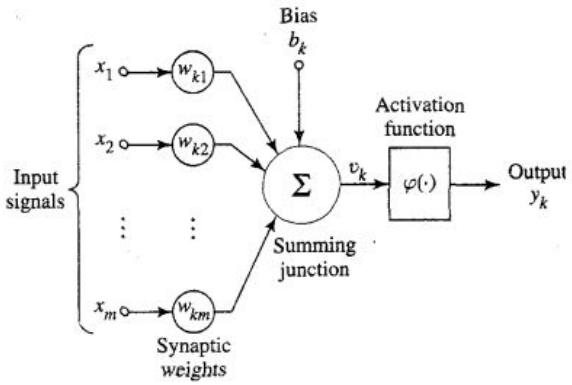


Figure 2.5: Visualization of the essentials parts of the ANNs, the neurons. This nonlinear model image is taken from [24].

2.2.2 Convolution layer

Convolution is a mathematical term frequently used in computer vision methods, which is defined as applying a function repeatedly over some array of values. From a computer vision standpoint, it means applying a filter over an image at all possible offsets. Having a small 2D image patch as input, the filter consistant of a layer of connection weights produces a single unit output. A convolution layer is parametrized by:

- Number of maps M,
- size of the maps (M_x, M_y),
- kernel sizes (K_x, K_y),
- skipping factors S_x and S_y that define how many pixels the kernel skips in both x and y directions between subsequent convolutions.

Each layer has the same number of maps and their size is equal. A kernel of size (K_x, K_y) is shifted over the image just like the sliding window technique. The kernel must be completely inside the image to be able to perform the convolution. The output map size is then defined as

$$M_x^n = \frac{M_x^{n-1} - K_x^n}{S_x^n + 1} + 1; \quad M_y^n = \frac{M_y^{n-1} - K_y^n}{S_y^n + 1} + 1, \tag{2.15}$$

where n is the index of the layer, K_x^n and K_y^n are the kernel sizes along both x and y directions in the n^{th} layer and finally S_x^n and S_y^n are the skipping factors in layer n along both x and y axes. Each map in layer L^n is connected to at most M^{n-1} maps in the layer L^{n-1}. The neurons of a particular map share their weights but have different input fields. The function of the convolution operators is to extract various features of the input. The first convolution layers obtain the lowest-level features such as lines, edges and corners. In contrast, the deeper layers obtain higher-level features.

2.2.3 Subsampling layer

The main role of the subsampling layer (or pooling layer) is to reduce the variance of the outputs from the convolution layers. Subsampling (or down-sampling) in general means reducing the overall size of a signal. Subsampling has a different meaning in different domains and in the domain of 2D filter outputs it can be understood as a method of increasing the position invariance of the filters (weights). This layer computes the maximum (like the popular LeNets in Caffe [28] do) or the average value of a particular feature throughout a region of the image. A very good characteristic of these computations for object detection and classification is that they are invariant to small translations.

To be more precise, the matrix of filter outputs is divided into small overlapping grids which take the maximum or average value in each grid as the value in the reduced matrix. Intuitively, the larger the grid - the greater the signal reduction. Oftenly, skipping factors are used to skip nearby pixels prior to convolution.

Consequently, using subsampling layers in between convolution layers increases the feature abstractness, resulting in local spatial abstractness increase as well. Therefore, CNNs take into account local information as well.

2.2.4 ReLU layer

ReLU (Rectified Linear Units) is a layer of neurons that use a non-saturating activation function

$$f(x) = max(0, x). \tag{2.16}$$

This layer increases the nonlinear properties of the decision function and the whole network, surprisingly without negatively affecting the receptive fields of the convolution layer.
Besides the *max* activation function, there are many other functions used to increase the nonlinearity, such as the ones mentioned in [30]:

- Saturating hyperbolic tangent $f(x) = tanh(x)$,
- absolute saturating hyperbolic tangent $f(x) = |tanh(x)|$,
- sigmoid function $f(x) = (1 + e^{-x})^{-1}$.

Compared to the aforementioned nonlinear functions, the ReLU activation function is easier to compute, thus greatly increases the neural networks training time. For example, in [30] two equivalent networks were trained with the *tanh* function and a ReLU function, and the ReLU reached the results six times faster. They also suggest that fast learning has a great influence on the performance of large models trained on large datasets. ReLUs have a very desirable characteristic that they do not require input normalization to prevent them from saturating. Even if a small portion of the training examples produce a positive input to a ReLU, learning will happen in that particular neuron. This only consolidates this layers usefulness in CNNs.

2.2.5 Fully connected layer

We have reached the high-level reasoning part of the neural network. After several convolution layers and max pooling layers, the neurons from the previous layer need to be connected to the fully connected layers' neurons, which is the main

purpose of this layer. These layers can be visualized as one-dimensional since they are not spatially located anywhere. That means that there can not be any further convolution layers after a fully connected layer. One good characteristic of these layers is that they work with multiple-dimensional neural networks. To understand the aforementioned layers better, Figure 2.6 shows how they are connected.

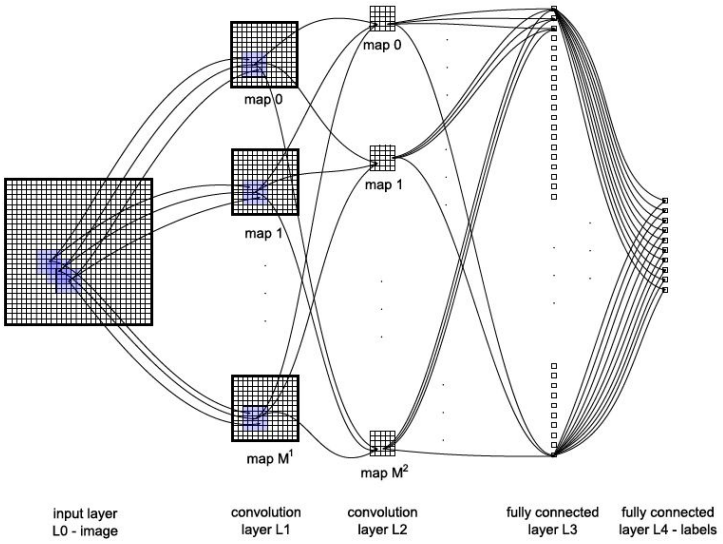


Figure 2.6: A visualization of the architecture of a CNN. Image taken from [8].

2.2.6 Error propagation in a fully-connected network

Having mentioned and described the types of layers a CNN is made of, it is time to have a look at the way the network learns and adjusts the neuron weights. To understand this, it is needed to know the basic types of neurons in a standard fully-connected neural network with L layers:

- An input layer (consisted of units u_i^0) whose values are fixed by the input data.
- Hidden layers (consisted of units u_i^ℓ) whose values are derived from their previous layers.

- Output layer (consisted of units u_i^L) whose values are derived from the last hidden layer.

The way a neural network learns is by adapting and correcting the weight w_{ij} of each unit u_i^ℓs output to some other unit $u_i^{\ell+1}$. Most commonly the learning is driven by a loss function (also commonly known as an error function or a cost function), by mapping many parameter settings with a goal to minimize the loss. An example of a popular loss function is the mean-squared error, i.e.,

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2, \tag{2.17}$$

where $\hat{Y}_i$ is a vector of n predictions, and Y is a vector of the observed values. This function is an average of the squares of the errors. Another common error function is the Euclidean distance function, i.e.,

$$E = \frac{1}{2n}\sum_{i=1}^{n}||\hat{Y}_i - Y_i||^2. \tag{2.18}$$

The loss is oftenly computed by performing a so-called forward propagation (FP).

2.2.7 Forward propagation in ANNs

In order to understand the training in networks with more sophisticated architecture like CNNs, first one needs to understand how the training is performed in the ANNs. The mathematical background is the same, but the derivations are slightly harder.

Training an ANN consists of iterating and computing the loss with a forward pass, then altering the weights on the output units based on this error and finally calculating the error in the hidden nodes. The output from a neural network is the last layer of units u^L. To be able to compute the output y_i^ℓ of unit u_i^ℓ (ith unit in layer ℓ) from its input x_i^ℓ it is needed to apply some nonlinearity $\sigma(x)$ to the input. The input to layer $\ell = 1$ is set upon initialization and the rest of the inputs of the following layers are computed as weighted sums of the outputs of the previous layer. So, the forward propagation algorithm looks like this:

1. Compute the activations for layers with known inputs:

$$y_i^\ell = \sigma(x_i^\ell) + I_i^\ell. \tag{2.19}$$

2. Compute the inputs for the next layer from these activations:

$$x_i^\ell = \sum_j w_{ji}^{\ell-1} y_j^{\ell-1}. \tag{2.20}$$

3. Repeat steps 1 and 2 until the output layer is reached and its values y^L are known.

When a unit has no inputs, the $\sigma(x)$ in unit y_i^ℓ becomes a constant, thus allowing to be set externally with the variable I_i^ℓ. As mentioned before, this is done at the input layer. First we set the activations in the input layer, then compute the inputs of the neurons in the following layer, use the nonlinearity to get their activations and propagate values until reaching the final layer. Then, the final activations y^L are computed. While propagating the values, the error function $E(y^L)$ is regularly calculated in order to estimate the networks quality. Like mentioned before, this loss function can be mean-squared error (MSE), Euclidean distance or many others. One constraint is that the derivative $\frac{dE(y^L)}{dy_i^L}$ depends only on y_i^L. This means that this error function must be computed per output and summed, resulting in heavy computation and ruling out complex functions which only makes the learning harder.

2.2.8 Backpropagation in ANNs

Once the forward propagation is done, the loss functions gradient needs to be computed by knowing the desired output for each input value. The main goal is optimizing the weights in order to minimize the error.
Backpropagation, an abbreviation for "backward propagation of errors", is the most commonly used technique in order to train the network, used with an optimization method such as gradient descent. The gradient descent can be Stochastic

gradient descent [4], Adaptive gradient [38] and many others such as Nestorov acceleration gradient [37]. Backpropagation is a generalization of the delta rule (which applies only to single-layered neural networks) by using the chain rule to iteratively compute gradients in each layer.
For further clarification, a chain rule is a formula in calculus for computing the derivative of a composition of two or more functions. The simplest form of the chain rule for two functions f and g is

$$(f \cdot g)' = (f' \cdot g) \cdot g'. \tag{2.21}$$

It is a necessity for the activation function used by the neurons to be differentiable. To be able to use gradient descent (or a different algorithm) to train the network, the derivative of the error needs to be computed with respect to each weight. Before digging deeper, it is crucial to remind the reader that derivatives are used because they measure the sensitivity to a change of quantity. Derivatives are also popularly described as "instantaneous rate of change". Using the chain rule, we get

$$\frac{\partial E}{\partial w_{ij}^{\ell}} = \frac{\partial E}{\partial x_j^{\ell+1}} \frac{\partial x_j^{\ell+1}}{\partial w_{ij}^{\ell}}. \tag{2.22}$$

The chain rule can be rewritten as

$$\frac{\partial E}{\partial w_{ij}^{\ell}} = y_i^{\ell} \frac{\partial E}{\partial x_j^{\ell+1}}, \tag{2.23}$$

since we know that the partial derivative with respect to the weight is the activation from its origin neuron. The y values are already known since they are calculated in the forward propagation phase, so what is left is the partial derivatives with respect to x_js, i.e.,

$$\frac{\partial E}{\partial x_j^{\ell}} = \frac{\partial E}{\partial y_j^{\ell}} \frac{\partial y_j^{\ell}}{\partial x_j^{\ell}} = \frac{\partial y_j^{\ell}}{\partial x_j^{\ell}} \frac{\partial}{\partial x_j^{\ell}} \Big(\sigma(x_j^{\ell}) + I_j^{\ell} \Big) = \frac{\partial y_j^{\ell}}{\partial x_j^{\ell}} \sigma'(x_j^{\ell}). \tag{2.24}$$

The only equation left to derive is the derivative with respect to the activation y_i^{ℓ}. The partial derivative and has the following two cases

$$\frac{\partial E}{\partial y_i^\ell} = \begin{cases} \frac{d}{dy_i^L} E(y^L) & if\ \ell = L \\ \sum \frac{\partial E}{\partial x_j^{\ell+1}} \frac{\partial x_j^{\ell+1}}{\partial y_i^\ell} = \sum \frac{\partial E}{\partial x_j^{\ell+1}} w_{ij} & if\ \ell \neq L \end{cases}. \tag{2.25}$$

After applying the chain rule, what we finally get is derivatives of the inputs to the next layer weighted by how much y_i^ℓ contributes to each input. Consequently, this means that the loss at a node in layer ℓ is the combination of losses at the next nodes in layer $\ell + 1$, weighted by the size of the contribution of the node in layer ℓ to each of those nodes in layer $\ell + 1$.
Having derived all of the equations, the whole BP algorithm for fully connected networks is:

1. Compute errors at the output layer L,
$$\frac{\partial E}{\partial y_i^L} = \frac{d}{dy_i^L} E(y^L). \tag{2.26}$$
2. Compute partial derivative of error with respect to neuron input (or the "deltas") at first layer ℓ that has known errors,
$$\frac{\partial E}{\partial x_j^\ell} = \sigma'(x_j^\ell) \frac{\partial E}{\partial y_j^\ell}. \tag{2.27}$$
3. Compute errors at the previous layer,
$$\frac{\partial E}{\partial y_i^\ell} = \sum w_{ij}^\ell \frac{\partial E}{\partial x_j^{\ell+1}}. \tag{2.28}$$
4. Repeat steps 2 and 3 until deltas are known at all but the input layer.
5. Compute the gradient of the error,
$$\frac{\partial E}{\partial w_{ij}^\ell} = y_i^\ell \frac{\partial E}{\partial x_j^{\ell+1}}. \tag{2.29}$$

Now that we have seen how the forward propagation and BP work, it will be easier to understand the same processes in CNNs. As shown in Figure 2.7, this CNN has three types of layers which all require slightly modified equations for both FP and BP. We have already derived the BP and FP equations for the fully connected part, so what is left is to derive them for the convolution layers and the sub-sampling layers.

2.2.9 Error propagation in Convolution layer

In order to compute the pre-nonlinearity input to some unit x_{ij}^{ℓ} in layer ℓ, it is needed to sum the contributions from the cells in the previous layer and weight them,

$$x_{ij}^{\ell} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \omega_{ab} y_{(i+a)(j+b)}^{\ell-1}, \tag{2.30}$$

where the filter ω is of size $m \times m$. Supposing we have a neuron layer of size $N \times N$, the output of the convolution with the respected filter ω will have size $(N-m+1) \times (N-m+1)$. Note that this is just an ordinary convolution operation. Furthermore, the convolution layer must apply its nonlinearity

$$y_{ij}^{\ell} = \sigma(x_{ij}^{\ell}). \tag{2.31}$$

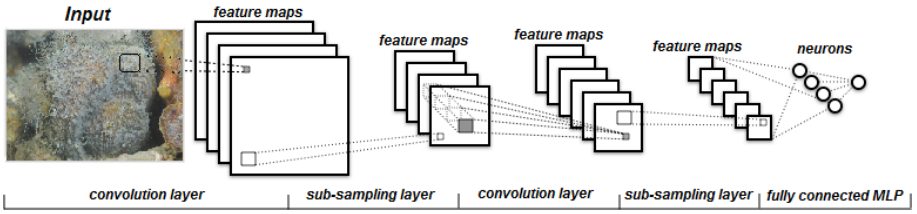


Figure 2.7: The LeNet architecture.

Assuming we already have an error function and the error values at the convolution layer are already computed through FP, we are interested in the error values in the previous layer and the gradients for all weights in the current layer. The gradients are computed by yet again applying the chain rule

$$\frac{\partial E}{\partial \omega_{ab}} = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\partial E}{\partial x_{ij}^{\ell}} \frac{\partial x_{ij}^{\ell}}{\partial \omega_{ab}} = \sum_{i=0}^{N-m} \sum_{j=0}^{N-m} \frac{\partial E}{\partial x_{ij}^{\ell}} y_{(i+a)(j+b)}^{\ell-1}. \tag{2.32}$$

As the derivation of the previous chain rules, there is a substitution applied from the FP equation ($\frac{\partial x_{ij}^{\ell}}{\partial \omega_{ab}} = y_{(i+a)(j+b)}^{\ell-1}$). Furthermore, the "deltas" ($\frac{\partial E}{\partial x_{ij}^{\ell}}$) are needed

to be computed by applying the chain rule once more

$$\frac{\partial E}{\partial x_{ij}^{\ell}} = \frac{\partial E}{\partial y_{ij}^{\ell}} \frac{\partial y_{ij}^{\ell}}{\partial x_{ij}^{\ell}} = \frac{\partial E}{\partial y_{ij}^{\ell}} \frac{\partial}{\partial x_{ij}^{\ell}} (\sigma(x_{ij}^{\ell})) = \frac{\partial E}{\partial y_{ij}^{\ell}} \sigma'(x_{ij}^{\ell}). \tag{2.33}$$

The deltas are easily computed since the current layers error is known and what remains is the derivative of the activation function $\omega(x)$.
The only thing left is to compute the weights for the convolution layer, by using the chain rule and propagating the errors to the previous layer

$$\frac{\partial E}{\partial y_{ij}^{\ell-1}} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \frac{\partial E}{\partial x_{(i-a)(j-b)}^{\ell}} \frac{\partial x_{(i-a)(j-b)}^{\ell}}{\partial y_{ij}^{\ell-1}} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} \frac{\partial E}{\partial x_{(i-a)(j-b)}^{\ell}} \omega_{ab}, \tag{2.34}$$

where the substitution from the forward propagation is $\frac{\partial x_{(i-a)(j-b)}^{\ell}}{\partial y_{ij}^{\ell-1}} = \omega_{ab}$. The equations looks similar to the one for convolution and is valid for points at distance higher than m of the top and left corners. This presents a problem in implementation and is usually solved by adding zeros along the edges of the matrix and applying this equation afterwards.
Having finished the theory behind the FP and BP in the convolution layers, it is very important to mention why there is not a separate subsection for FP and BP in subsampling layers. It is because the subsampling layers do not do any learning by themselves. Like mentioned before, their job is just to take a region of size $k \times k$ from a $N \times N$ big layer and output $\frac{N}{k} \times \frac{N}{k}$ layer with maximums of the regions, that is single values. Then, these values acquire errors that are computed by BP from the previous layer and are later forwarded from the place they came from.

This paragraph concludes the theory behind these extraordinary ML methods. The chapter is finished by a couple of final notes.
An improved version of the typical feed forward neural network architecture is shown in Figure 2.8, which shows that during BP it is only needed to adjust the number of parameters equal to a single instance of the filter. Theoretically it is possible to add more filters on top of the output from the previous filter bank. However, this is proven to be useless because the dimensionality of applying a filter is equal to the input dimensionality. It means that these additional filters are

not invariant to translation, which needs to be solved with another type of layer, a subsampling layer. Finally, some researchers [7] even suggest to ℓ^2 normalize the CNN features before using SVM for improved classification. Equation 2.35 represents the ℓ^2 normalization step of $\mathbf{x}$, where $|\mathbf{x}|$ denotes the normalized vector.

$$|\mathbf{x}| = \frac{\mathbf{x}}{\sqrt{\sum_{k=1}^{n} |x_k|^2}} \tag{2.35}$$

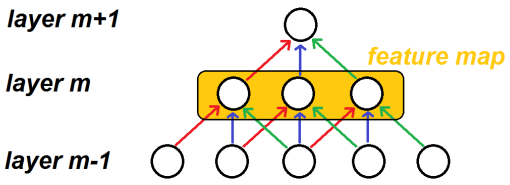


Figure 2.8: The sparse convolution connectivity between neurons weights which has a receptive field behavior.

2.3 Support vector machines

Support Vector Machines (SVM) [12] are very popular models for classification and regression. In machine learning, a classification problem consists of data belonging to a certain class. Different models are trained on a small portion on this data and later are validated on the remaining examples. SVM models are extensively and frequently used because of numerous good characteristics such as resiliency against overfitting, no local minima, effectiveness in high dimensional spaces, memory efficiency and adaptability courtesy of the variety of kernel functions.

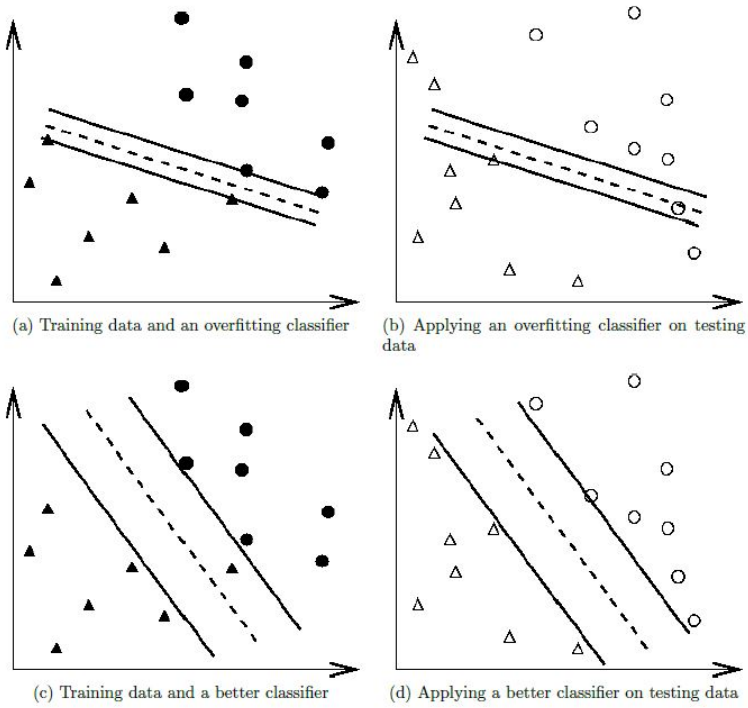


Figure 2.9: Example of one of the advantages of SVM, resistant against overfitting. The subplots show an evident difference between an overfitting and a non-overfitting model. Image taken from [6].

In a binary classification problem the support vector machines task is to find a N-1 dimensional hyperplane (or a set of hyperplanes) in a N-dimensional space that separates the classes as accurately as possible, like in Figure 2.10. The quality of the separation is measured by the distance from the hyperplane to the nearest training data points of the two classes (the function margin). Given a data of n points which have assigned class labels $y_i \in \{-1, 1\}$, we are looking for the hyperplane defined by a set of points $\mathbf{x} \in \mathbb{R}^N$ that satisfies the equation (2.36), i.e.,

$$\mathbf{w}^T \cdot \mathbf{x} - b = 0, \tag{2.36}$$

where $(\cdot)$ denotes the dot product, $\mathbf{w}$ is a normal vector to the hyperplane and b

is the offset. Linearly separable data have two hyperplanes defined as

$$\begin{aligned} \mathbf{w}^T \cdot \mathbf{x} - b &= 1, \\ \mathbf{w}^T \cdot \mathbf{x} - b &= -1. \end{aligned} \tag{2.37}$$

In this case, the distance between the two of them is $\frac{2}{\|\mathbf{w}\|}$ and further minimization of $||\mathbf{w}||$ is required. The minimization is done with the following equation and its constraint

$$\arg\min_{w,b,\xi_i} \frac{1}{2}\|\mathbf{w}\|^2, \quad \text{subject to } y_i(\mathbf{w}\cdot x_i - b) \geq 1 \tag{2.38}$$

The constraint is necessary in order to prevent the data points falling into the margin. In (2.38) $||\mathbf{w}||$ is replaced with $\frac{1}{2}||\mathbf{w}||^2$ in order to speed up the computation without changing the solution.

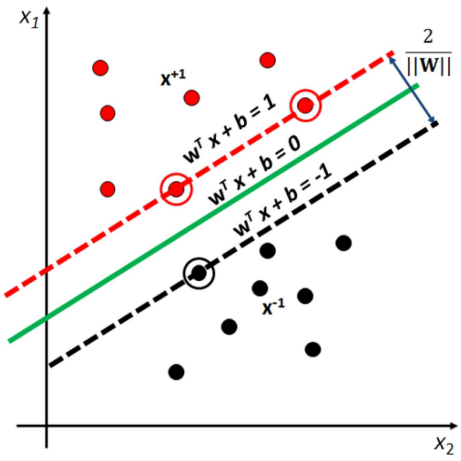


Figure 2.10: A binary linearly separable SVM problem illustrated on two-dimensional space. The black and red dotted lines represent the support vectors, the green line is the hyperplane and the colored circles are the two classes, respectively.

Ideally, the data is separated such that on each side of the hyperplane the class labels belong to the same class. This type of hyperplane is known as the maximum-margin hyperplane. Figure 2.10 illustrates this type of hyperplane, which is the

green line. However, in realistic problems this type of separation happens extremely rarely.

Because of the rarity of the aforementioned perfectly separable examples, the equations need to be reformulated and made more powerful to properly handle the misclassified examples. In 1995 slack variables were introduced [12] that measure the extent of missclassification. The reformulated optimization problem becomes

$$\begin{aligned} \arg\min_{w} \quad & \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}(\xi_i), \quad \text{subject to } y_i(\mathbf{w}\cdot x_i - b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, C \geq 0. \end{aligned} \tag{2.39}$$

This is known as the *primal* problem. The variable C is the penalty parameter that controls the trade-off between the margin maximization and the misclassification error. It is set by an expert. A common practice [27] is calculating it using a grid search or an even smarter alternative is performing a five-fold cross validation on the training set [18].

The SVM can be formulated to learn a linear classifier by solving a so-called *dual* optimization problem, i.e.,

$$\begin{aligned} \arg\max_{\alpha_i \geq 0} \quad & \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j k(x_i, x_j); \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C \text{ for } \forall i, \\ \text{and} \quad & \sum_{i=1}^{n}\alpha_i y_i = 0, \end{aligned} \tag{2.40}$$

where the α_i are the "dual" variables. The main advantage of a linear penalty function is that the slack variables vanish in the dual problem. Many of the α_i become zero in practice, the others correspond to the weights of the corresponding training data-points, which are called the support vectors.

The results obtained in this thesis are using a SVM classifier with $L2$-regularized Logistic Regression [19]. This SVM is the solution of the following unconstrained optimization problem

$$\arg\min_{w} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n}\log(1+e^{-y_i w^T x_i}), \tag{2.41}$$

The corresponding *dual* form is

$$\begin{aligned} \arg\min_{\alpha} \quad & \frac{1}{2}a^T Q\alpha + \sum_{i:\alpha_i>0}\alpha_i\log\alpha_i + \sum_{i:\alpha_i<0}(C-\alpha_i)\log(C-\alpha_i) - \sum_{i=1}^{n} C\log C, \\ \text{subject to} \quad & 0\leq\alpha_i\leq C, i=1,\ldots,n, \\ \text{where} \quad & Q_{ij} = y_i y_j x_i^T x_j. \end{aligned} \tag{2.42}$$

This classifier was used to get probabilistic outputs in addition to the classification. For better understanding of the following contents of this chapter, we need to define what a kernel is and explain its significance.
Kernel is a shortcut that allows us to do certain calculations faster which otherwise would involve computations in higher dimensional space by using kernel functions. Figure 2.11 explains the intuition behind the kernel functions. In this case, no 1-dimensional hyperplane can separate the data of the two classes shown in the upper part of the figure. However, if the data was to be somehow projected in a 2-dimensional space, in this case done by $x \to (x, x^2)$, we would be able to separate the data.

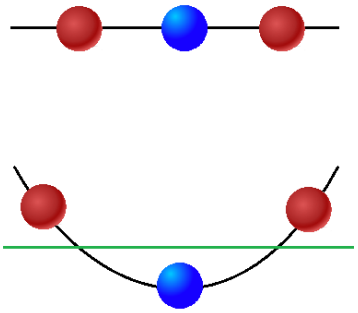


Figure 2.11: 1 dimensional example of kernel functions.

The kernel functions role is to operate in a high-dimensional, implicit feature space without needing to compute the coordinates of the data in that space, but rather by computing the inner products between all pairs of data in the feature space. This approach is called the "kernel trick" and is often computationally cheaper than explicitly computing the coordinates.
In the following examples of non-linear kernels the parameters are γ, r and d:

- Radial Basis Function (RBF): $K(x_i, x_j) = e^{-\gamma||x_i - x_j||^2}, \gamma > 0$,
- Polynomial: $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d, \gamma > 0$,
- Sigmoid: $K(x_i, x_j) = \tanh(\gamma x_i^T x_j + r)$.

One advantage of a linear SVM kernel is that it requires only C to be set, while non-linear SVM kernels require setting multiple parameters.
In object detection problems, it is of vital importance to know the degree of certainty of an example being classified in the assigned class. These probabilistic outputs are obtained using the Platt Scaling (or Platt calibration) [39]. The probability that the example belongs to the class $y = 1$ is

$$P(y = 1|x) = \frac{1}{1 + e^{Af(x)+B}} \tag{2.43}$$

where A and B are scalar parameters that are learned by the algorithm. It is a logistic transformation of the classifier scores $f(x)$. Using elementary statistics and the knowledge that this is a binary classificatin problem, we know that the probability that the example belongs to the class $y = -1$ is

$$P(y = 0|x) = 1 - P(y = 1|x) \tag{2.44}$$

It is not obligatory for hyperplanes to pass through the origin of the coordinate system. The ones that pass the origin are called unbiased and those that do not are called biased.

One final note is that the support vector machines with lower C acquire a solution faster than the ones with big C. This happens because C defines the amount of outliers that are taken into account in calculating the support sectors.

For large values of C, the optimization will look for a smaller-margin hyperplane aiming at classifying training points correctly. Sometimes, this is not desired, because higher-margin hyperplanes generalize the problem better. Consequently, a high cost value C forces the SVM to create a complex prediction function and usually overfit the problem, while a lower cost parameter C leads to a simpler prediction function.

Chapter 3

Proposed approach

The solemn goal of this thesis was to develop a method that is able to count densely populated objects, in particular polyps. In order to do so, we had chosen several state-of-the-art machine learning methods that have already achieved excellent results in some computer vision domains. Our proposed pipeline of algorithms is shown in Figure 3.1. This simple diagram is used for the reader to better understand how our object detection algorithm works.

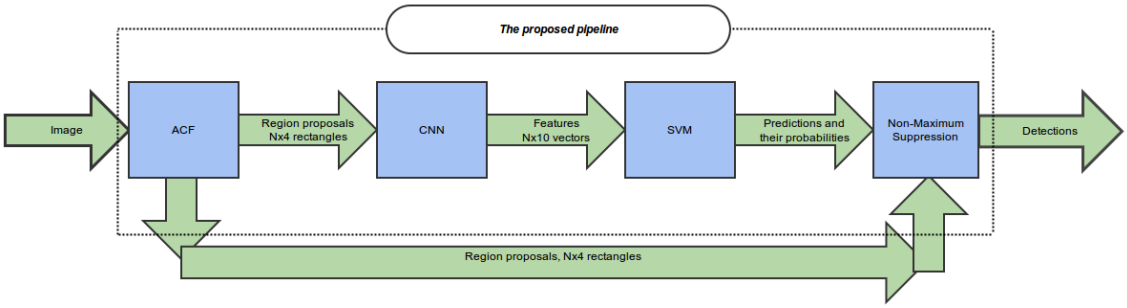


Figure 3.1: Visualized pipeline of algorithms that is used to detect the polyps. First an image is used as an input to ACF [16] that outputs region proposals, which are then having their features extracted by a CNN [30]. The features are then labeled by an SVM classifier and further filtered by a non-maximum suppression, resulting in detections.

3.0.1 Region proposals

A crucial topic that is being constantly improved in object detection is object proposals [1, 47, 48]. Given an image, we are interested which areas are likely containing the desired objects. Until recently, object detection solutions were using the multi-scale sliding window technique to naively search the entire image for objects [49, 17, 45]. Consequently, a classifier needed to classify each of those windows, leading to a lot of potential False Positives (regions that are falsely labeled as positive). This problem has lead researchers to develop smarter ways to search the image for potential objects. In [1] a lot of different methods that propose regions are evaluated. The region proposals returned from these clever methods are a small subset of the ones that a sliding window would return. The significance of these methods is that we usually save computational time and avoid a lot of False Positives. Accordingly, we can use this computational time to apply an even stronger classifier to these regions.

We chose to work with the aggregated channel features (ACF) detector because of the appealing results it has achieved [16]. This detector is thoroughly described in Section 2.1. It takes the image as an input and outputs regions that potentially have objects of interest in them. The challenge with ACF was to train a model whose outputs cover as much ground truths as possible, while not proposing too many regions so that the later classification algorithm would have to be extremely powerful and do all the work. The training should be performed by a careful selection of parameters and channels. In some domains the channels might not be as informative as in others. That is why a visualization of these channels is important. Other than that, the non-maximum suppression parameter has to be precisely set depending on the size of the annotations. A bad non-maximum suppression threshold leads to loss of valuable regions and in the overall performance of the method. In Section 6.2 of [16] there are further constructive insights about parameter selection and the way they have tackled this problem.

3.0.2 Feature extraction by CNN

Having obtained the regions that might be the desired objects, object detection solutions need to obtain insight feature information about them.
The second step of our approach is convolutional neural network (CNN) features extraction. Features are arguably the most important part of any classification problem. Having all of the good characteristics mentioned in Section 2.2 and the recent success of researchers [30, 8, 7] that have used this method, the CNNs appear well suited for the task ahead. One of the disadvantages, which is tedious amount of training time and hardware requirements was avoided by using an already trained CNN, whose structure is shown in Figure 3.2.

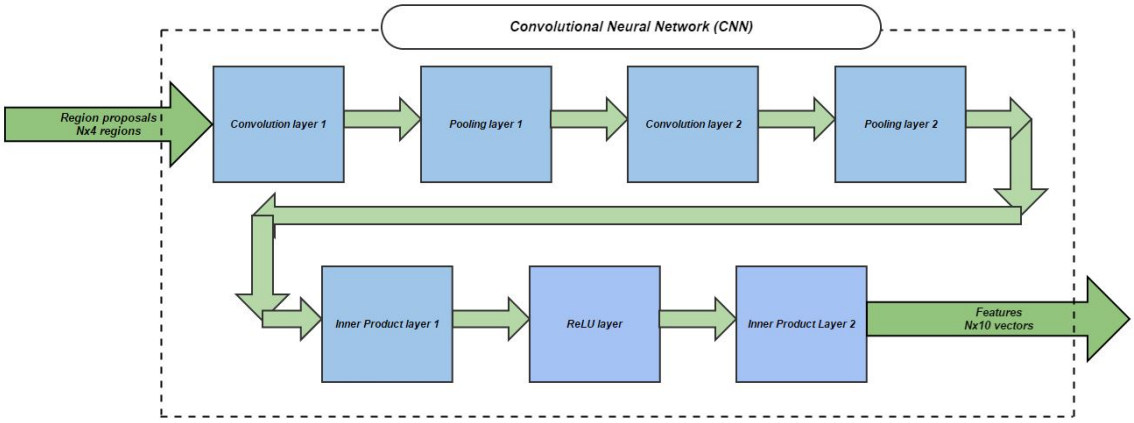


Figure 3.2: The CNN and its 7 layers made up from two convolution layers followed by pooling layers, then there are two InnerProduct layers (commonly known as fully connected layers) with a ReLU between them.

The original CNN in Caffe [28] had a SoftMax layer at its end, which was redundant in our case. The parameters used in each layer are described in detail in Section 4.2. An important aspect of this CNN is the Xavier algorithm, which is implemented based on the paper [23] and initializes the networks weights. This initialization step is crucial since the Xavier initialization helps the signals reach deep into the network. If the weights in a network start too small, then the signal shrinks as it passes through each layer until it is too tiny to be useful. In contrast,

if the weights are too large at initialization, the signal will grow so much when passing through each layer until the point it is too big to be of any use. This initialization method tries to make sure the initialization is just right, keeping the signal at reasonable range of values. In Caffe, the initialization of the weights is based on a distribution with $\mu = 0$ and some specific variance

$$Var(W) = \frac{1}{n_{in}}, \tag{3.1}$$

where W is the initialization distribution (which is typically Gaussian or uniform) for the neuron and n_{in} is the number of neurons feeding into it. More details about the initialization step can be found in [23].

3.0.3 SVM application

After obtaining characteristics about the regions, it needs to be determined whether they are polyps or not. The L2-regularized Logistic Regression SVM classifier classified the examples and obtained probability estimates which indicate the level of certainty that the classification was correct for each example. The theory behind this model and how the probability outputs are obtained is covered in Section 2.3 after the theory for slack variables. Therefore, the outputs of the SVM were class labels for each instance and the likeliness each of them was classified accurately.

3.0.4 Non-maximum suppression

It is very important to note that the ACF did not necessarily output only one region proposal per object in the image. An object can have none or many region proposals. However, we would like to have only one rectangle detection per object. In order to achieve this, the final part of this pipeline is a non-maximum suppression. Similar to [22], this thesis is using an adroit non-maximum algorithm that greedily focuses on the more probable detections. It naively takes the probability distribution, sorts the data in an descending order and then starts iterating over the examples with the higher probabilities. If any rectangle B overlaps with the selected rectangle A by more than a certain threshold τ, then rectangle B is

discarded. The Overlap τ is measured by the ratio

$$\tau = \frac{A \cap B}{A \cup B}. \tag{3.2}$$

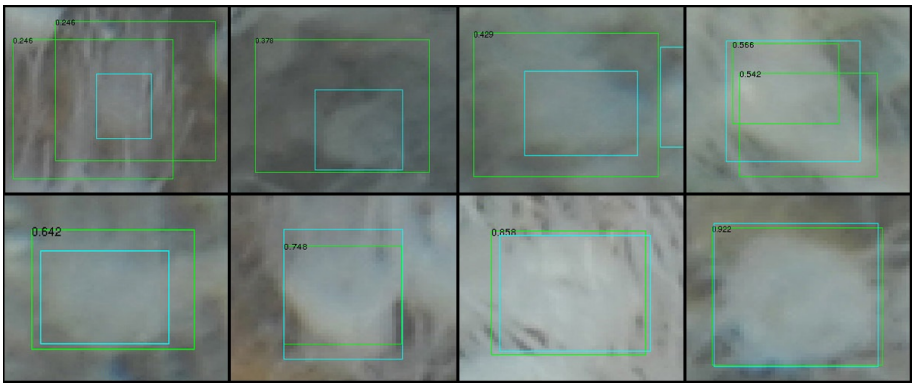


Figure 3.3: Examples of τ between two rectangles starting from the upper leftmost image with $0.2 \leq \tau \leq 0.3$ up to the lower rightmost image with $0.9 \leq \tau \leq 1.0$. The cyan rectangles are ground truth, while the green ones are region proposals.

The initial sorting is done to ensure that if such rectangle B is found, it is always going to have a probability score less than A. This non-maximum suppression method heavily relies on the probability distribution and takes advantage of it. Figure 3.3 aims to give the reader a feeling about the strength of the overlapping measure τ. Beginning with the upper leftmost image where the threshold is $0.2 \leq \tau \leq 0.3$ and ending with the lower rightmost image with threshold $0.9 \leq \tau \leq 1.0$, the thresholds in the images increase by 0.1. The upper row of images shows that it is difficult to determine whether a region belongs to an object when the region has $\tau < 0.4$.
To conclude this chapter, we provide a pseudocode for our pipeline of algorithms below in Algorithm 1.

Data: Image I;
`// Section 2.1 and Subsection 3.0.1`
for *window in multi-scale sliding windows* **do**
  Compute HOG, LUV and normalized gradient magnitude channels;
  Downsample the channels by summing and smoothing;
  Vectorize the channels;
  Apply a combined Decision trees and Adaboost classifier on the vectorized channel features;
  if *the window is classified as positive* **then**
    Keep the rectangles coordinates and its width and height;
  end
end
Suppress the very dense regions with non-maximum suppression;
`// Subsections 2.2.6-2.2.9, Subsection 3.0.2`
Do a forward CNN pass on the proposed regions and extract their features;
`// Section 2.3 and Subsection 3.0.3`
for *region in ACF proposed regions* **do**
  Project the features vector into the SVM space;
  if *the projected point lies on positive side of hyperplane* **then**
    Calculate probability of correct classification;
  else
    Discard the region;
  end
end
`// Section 2.3 and Subsection 3.0.4`
Sort the data proportionally to the probabilities;
for *region in sorted regions* **do**
  For each region calculate the overlap with all other regions;
  if *any* $\tau > threshold$ **then**
    Discard the corresponding regions;
  end
end
Result: Detections;

Algorithm 1: Our approach.

Chapter 4

Experimental results

Our approach proposed in the previous chapter was analyzed in detail to determine its quality and find out its strengths and weaknesses.
This structure of this chapter is the following: Section 4.1 describes the properties of the dataset in detail. Afterwards, Section 4.2 describes the implementation. Next, the parameters used for the algorithms are presented in Section 4.3. Following the parameter decisions, Section 4.4 contains the measures that were used to determine the quality of approach. Lastly, we present the results and their detailed analysis in Section 4.5 and draw some important conclusions.

4.1 Dataset

The dataset consists of 39 images of 44310 rectangle annotations. 35 of them were used for experimental results, made up from 39212 annotations, while the others were excluded because of their overly ambiguous regions and inappropriate quality. They were not omitted from the dataset because a lot can be learned about an algorithms behavior by testing on such images. Each pictures resolution is 4288×2848. Figures 4.1 and 4.2 are examples of images from the dataset.

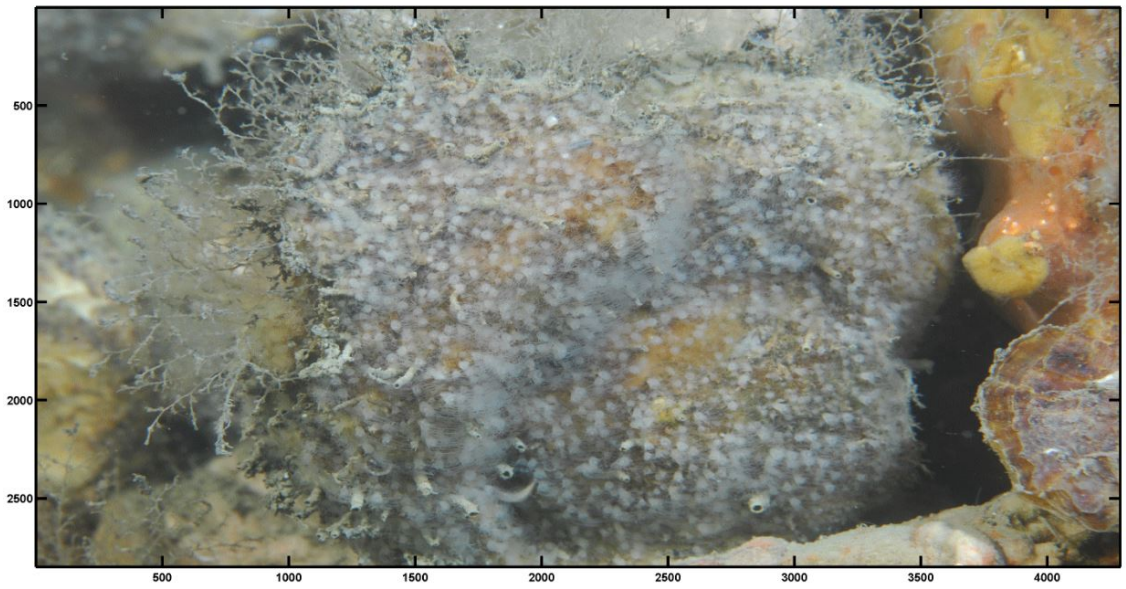


Figure 4.1: Example image 1 from the dataset.

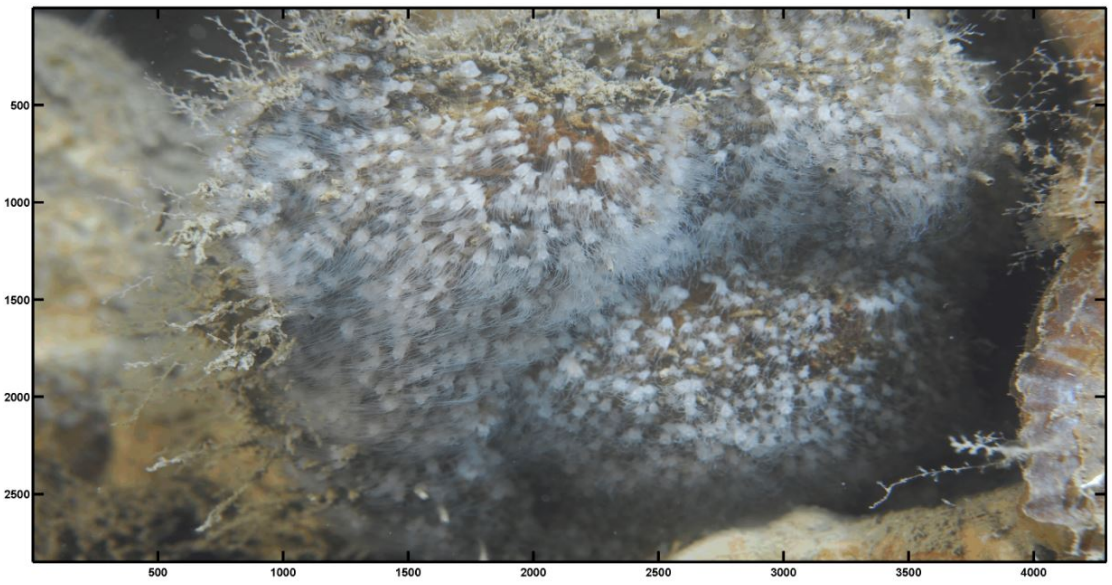


Figure 4.2: Example image 2 from the dataset.

Despite the patient annotating efforts, there are still some polyps within the

oysters area that are not annotated. Also, the images have polyps which are located around the oysters and are not annotated. The oysters are in main focus of the camera and everything around them is very blurry and ambiguous, where even humans would struggle to determine whether some parts of the image are polyps or not. Figure 4.3 illustrates how the polyps are distributed across the images. The heatmap is build such that all annotations increase the weights of the pixels where they are located. For example, if one annotation has $width = 5, height = 6$, then the 30 pixels covered by the rectangle get their weights increased by 1. Consequently, the brighter the region - the denser the population of polyps in those regions. Each image had on average $\mu = 1069$ annotations with standard deviation $\sigma = 288$.

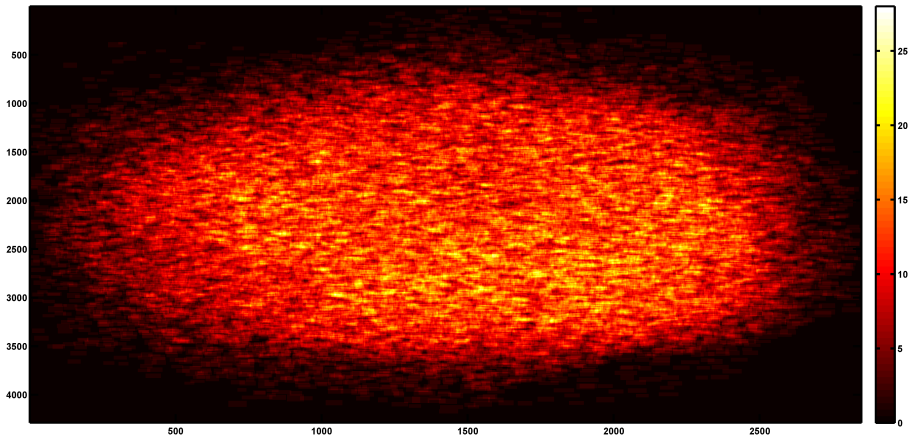


Figure 4.3: Distribution of the polyps across the images represented as a heatmap.

There are some polyp-like shapes and colors in the images. Some of the polyps are so densely distributed that they greatly overlap and occlude each other. This happens quite a lot and if there is low brightness in that particular part of the image, any kind of algorithm would undoubtedly struggle detecting these kinds of objects. Two typical kinds of noise are visualized in Figure 4.4. The polyps located on the left side of the image have higher brightness and are clearly separated from the background, while the polyps on the right side are blurry. There is also quite

a lot salt and pepper noise in each of the images, making the regions around the central oyster not suitable for experimenting. This problem is solved by annotating polygons for each of the images, indicating the area of the oysters. Our proposed pipeline was trained and tested only on the regions within the polygons.

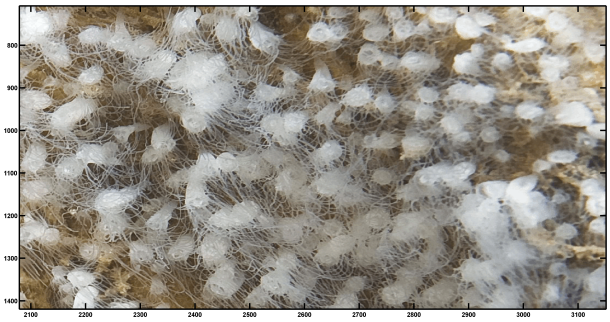


Figure 4.4: Visualization of a zoomed region of an image, characterized by occlusion and sharpness differences between parts of the image.

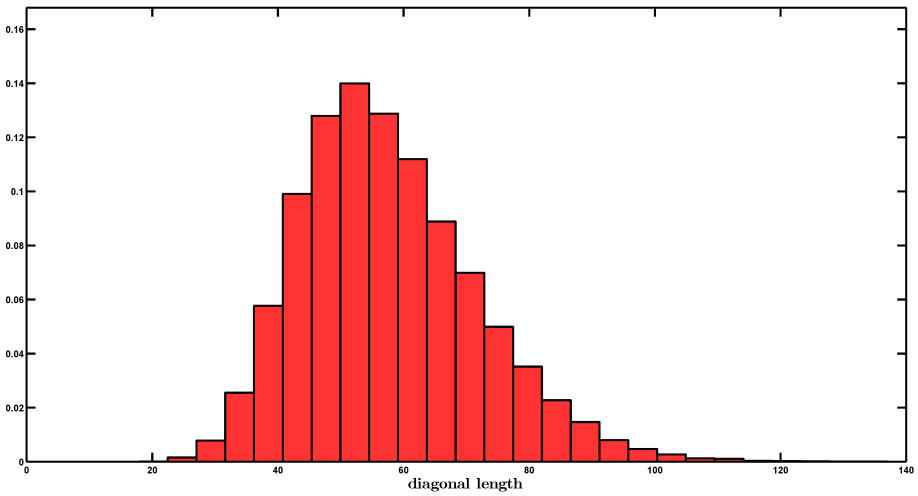


Figure 4.5: Histogram representing the diagonal length of the annotations.

Figure 4.5 shows some statistics about the annotations. An annotation had

a mean diagonal length of $\mu = 57$ with standard deviation $\sigma = 14$. It can be concluded from this histogram that the sizes of the annotations varies quite a lot.

4.2 Implementation details

Obtaining regions that may be the desired objects was quite challenging. To do so, the ACF implementation from P. Dollár [15] was used. There were some difficulties while training new ACF models, because some models were outputting many redundant regions, meaning that the parameters used for the training are fairly sensitive. Figure 4.6 shows the difference between two of the many trained ACF models.

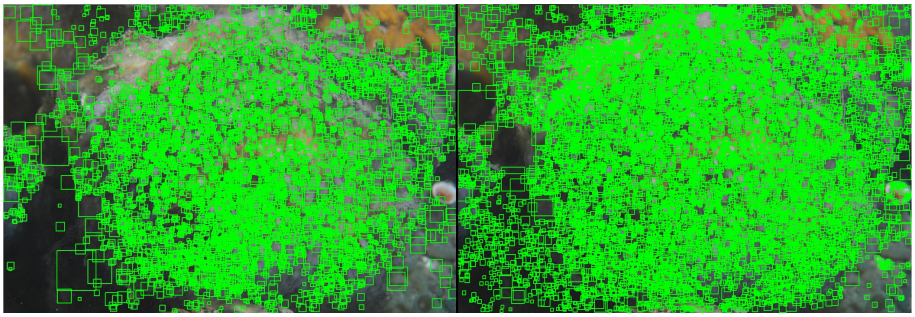


Figure 4.6: Visualization of the difference between two models trained with 1300 positive examples and the one on the right with more than 3000.

After any classification model has been trained with an increased amount of examples, it usually adapts to the problem better, resulting in superior results to the ones with less training examples. However, this is not always the case. We wanted to see whether increasing the training examples will over exaggerate the model. It is quite evident that the model on the left which was trained with considerably less positive examples performed better, since the one trained with more positive examples returned too many region proposals. This is probably due to the lack of expertise in setting the training parameters. ACF managed to return promising results which are analyzed in Section 4.5.

After obtaining the regions, an already trained CNN in Caffe [28] on the dataset MNIST was used for feature extraction. Its structure is shown in Section 3.0.2 on Figure 3.2 and the detailed parameters in Section 4.3.
The resulting CNN features were classified by the commonly used LIBSVM toolbox [6]. However, this toolbox proved to be quite slow and inefficient compared to LIBLINEAR [18]. The training phase of each kind of experiment in LIBSVM lasted around 6 minutes, while in LIBLIENAR only a few seconds. Since there is no optimized grid search for the best C parameter in the toolbox for MATLAB, the best C search was conducted through for loop repetition for some common C parameters resulting in huge time setback to obtain results. LIBLINEAR did the search for optimal C in a couple of seconds. In conclusion, LIBLINEAR was the better choice for this particular domain.

4.3 Parameter selection

This section explains the selection of the parameters used in the algorithms and equations.
The dataset was split into 5 images (containing 5572 annotations) for training and 30 (containing 33640 annotations) for testing, resulting in split ratio 14%:86%. The ACF model that was used was trained on 1300 annotations from two images, which is a small percentage (3.3%) of all of the annotations. There were 86 crops of images that were used as negative examples and the ACF was sampling 25 at most from each, making 2150 negative examples in total. Other than that, the stride used for the ACF was 4 and at each training stage there were 32, 128, 512 and 2048 weak classifiers, having model height and weight of size 30 along both axes respectively.

Our LeNet CNN [28] had its default parameters set. Each of the two convolution layers had convolutional kernel size 5, stride 1 and produced 20 output channels. On the other hand, the bias was initialized as constant, by default as 0. The weight learning rate was the same as the learning rate given by the solver during runtime and the bias learning rate was twice as big, which is believed by

Caffe [28] researchers to result in better convergence rates.
The pooling layer used kernels of size 2 and strides 2, which resulted in neighboring pooling regions that did not overlap. The first fully connected layer was using the same parameters and algorithms for learning and initialization, but had 500 outputs. The second fully connected layer, being the last layer in the network, produced 10 outputs (the final features). The original architecture of the LeNet had a Loss layer, however it was not used in our pipeline since it outputted results that had around 5% weaker Accuracy. This network was trained with 100 iterations.
Regarding the parameter selection of the classifier, finding the optimal C parameter was automatically done by LIBLINEAR [18, 19] by conducting a five-fold cross validation.

Nevertheless, choosing the ideal overlapping threshold parameter τ was difficult and required visual examining, which is shown in Figures 4.7 and 4.8.
We were interested whether the proposed regions of several different threshold ranges belonged to any of the polyps. Only after visually comparing the different threshold ranges we were able to determine which regions should be labeled as negative by defining the τ threshold. Figure 4.8 clearly shows that the second threshold (yellow colored rectangles), which is 0.2 and above is the ideal value for labeling the proposed regions. In particular, every region that had $\tau > 0.2$ was considered a positive example. The non-maximum suppression threshold was set to $\tau = 0.1$.

Lastly, we would like to mention the parameters for the ACF detector used as a baseline to compare our approach in Subsection 4.5.2. It was trained on nearly the same amount of parameters, however with much more discriminative negative examples. It was searching for objects on 54 different scales and used a 0.1 overlap parameter for its non-maximum suppression.

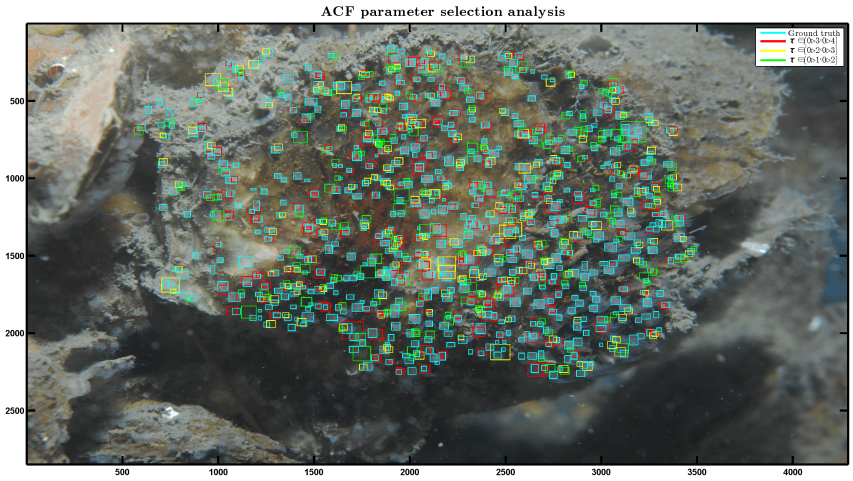


Figure 4.7: Visualization of three different value ranges for threshold τ.

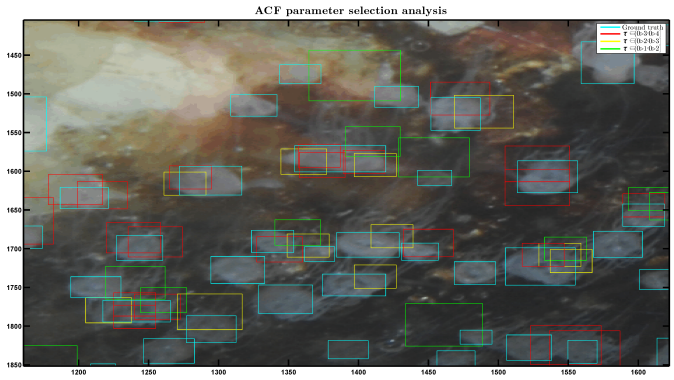


Figure 4.8: This figure is only a zoomed region of the previous Figure 4.7, clearly showing the differences between various thresholds.

4.4 Performance measures

The quality of the classification needs to be evaluated. In machine learning, this is usually done by calculating a confusion matrix (also known as contingency ta-

ble), which allows visualization of the performance of the classifier. As shown in Table 4.1, each row of the matrix represents the instances of the ground truth class and each column the instances of the predicted class. The most important abbreviations in the table are:

- TP - True Positives, polyps correctly classified as polyps.
- FP - False Positives (or type I error α), polyps incorrectly classified as non polyps.
- FN - False Negatives (or type II error β), non-polyps incorrectly classified as polyps.
- TN - True Negatives, non-polyps correctly classified as non-polyps.
- PPV - Positive predictive value (or Precision) is the fraction of the polyps that were correctly classified among all examples classified as polyps.
- TPR - True positive rate (or Recall, Sensitivity, hit rate) is the portion of the polyps that were correctly classified among all polyps.

Mathematical formulations of these performance measures are

$$Precision = \frac{TP}{TP+FP}, \quad Recall = \frac{TP}{TP+FN}, \quad Accuracy = \frac{TP+TN}{TP+FP+FN+TN}.$$

		Predicted class		
		Polyp	Non polyp	Total
Ground truth class	Polyp	TP	$FP(\alpha)$	PPV
	Non polyp	$FN(\beta)$	TN	NPV
	Total	TPR	FPR	N

Table 4.1: Confusion matrix.

4.5 Results

This section focuses on the results obtained by our approach. The first subsection presents the ACF results and analysis, following by subsection which is centered around the results before and after non-maximum suppression, as well as comparing our approach with another baseline algorithm. The third subsection is a visual representation of the quality of our detector. The final subsection presents analysis for potential future improvements.

4.5.1 ACF results and analysis

For the 35 images with 39212 annotations in total, the ACF detector returned 257944 regions of interest, 160043 of which were in the polygons. It basically means that on average were 4 times as much proposed regions as ground truths. Further analysis were carried out to measure the overall quality of the proposed regions returned by ACF that are shown in Figures 4.9 and 4.10. Figure 4.9 shows how good on average the proposed regions were. The main point of the histogram in Figure 4.9 is to demonstrate the level of overlapping accuracy the proposed regions possessed on average. We wanted to see how good on average will the regions labeled as positive overlap with the ground truths. The histogram was computed by splitting the proposed regions into 10 groups regarding their best overlap values τ with any annotation. The y axis represents the portion of regions in each of those groups. The other value above the histogram bins shows the mean value of the bins on its right (including that particular bin). Since the threshold τ for the positive examples was 0.2 and 0.3 for the experiments of the regions enlarged by 10% and more, the average quality of the proposed positive regions was 0.51(like shown above bin 2) and approximately 0.54 for the ones with $\tau > 0.3$. This analysis is only conducted on the proposed regions that have positive τ, since the ones that have $\tau = 0$ are a lot more than the positive ones and would dominate the histogram if they were shown. That means that the averages shown above the histogram bins are only calculated from the positive τ values.

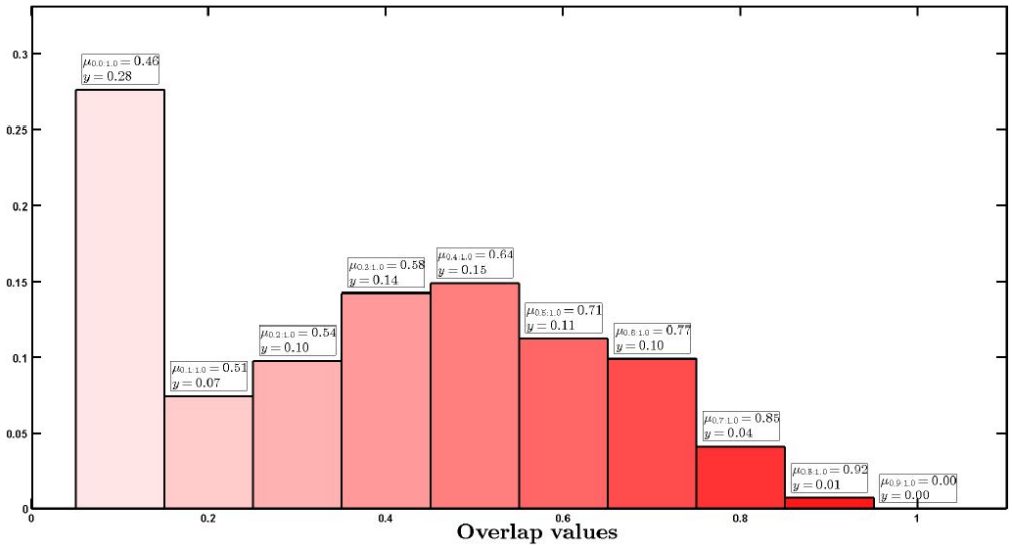


Figure 4.9: Analysys of the results from the first part of the pipeline, the ACF. The first row of the text above the bins shows the mean value of the bins on its right (including that particular bin). The second row is the value of the bin on the y axis.

In Figure 4.10 the bar plot shows the quality of the ACF regions proposal from another perspective. The bins represent the portion of annotations that would have a proposed region if the τ threshold was their previous x axis value. As mentioned before, the threshold used in most experiments was 0.2, which means that 96.4% (as shown above the third bin) of the annotations had a proposed region with $\tau > 0.2$.

We can conclude from Figures 4.9 and 4.10 that there are a lot of regions with $0 \leq \tau \leq 0.2$ as shown from the first bin in Figure 4.9. However, these regions are not problematic in reality as shown in Figure 4.10, because there are very few regions that have proposed regions with best τ below 0.2 around them. Only 3.2% of those annotations have a proposed regions with best τ in range $(0, 0.2]$, which means that non-maximum suppression should be able to handle the rest of the weak region proposals because they are located besides more powerful proposals.

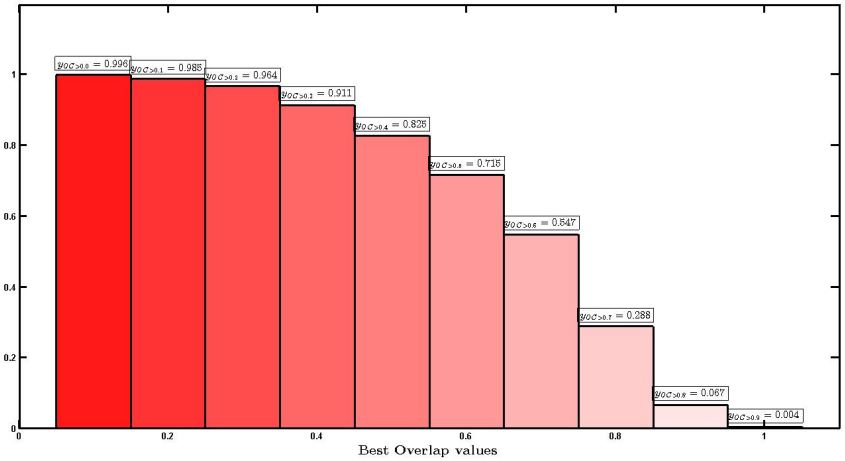


Figure 4.10: Bar plot showing the portion of annotations that have a proposed region with τ larger than their previous x axis values. The text above the bins represents that portion value.

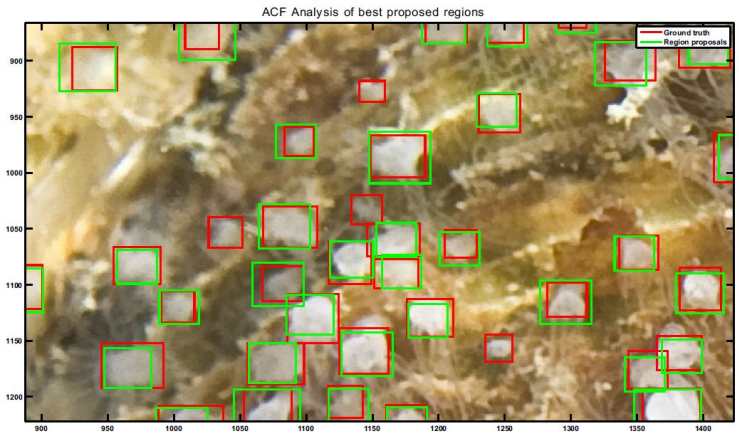


Figure 4.11: An example of the rare misclassifications of ACF. The red rectangles are the ground truths and the green ones are the region proposals. Next to each green rectangle there is a number showing its overlapping coefficient to a ground truth. The four red rectangles without a green rectangle besides them are the regions that ACF missed.

There are obviously a lot of annotations that have proposed regions of decent quality. We were very keen to know where the ACF detector missed to propose regions. Figure 4.11 has shown us that the vast majority of missed annotations are smaller than the rest or have abnormal shape.

4.5.2 Final results

This subsection covers the final results of our proposed approach on the newly annotated dataset. Four different experiments were conducted for both scaled and normal data, each exploring the idea of ℓ^2 normalizing the data as well. The intention behind every experiment was further improving our detector and learning more about the nature of our algorithms.

The first experiment was conducted using a classic statistics method which centers the data by subtracting the mean and dividing with the standard deviation. However, centering the data did not improve the detector. The rest of the experiments were increasing the size of the proposed regions by 5%, 10% and 15%. The main goal of these three experiments was finding out whether additional background information would help the detector to achieve more accurate detections. It turned out that the additional background information slightly improved our approaches accuracy and the model that increased the images proposed regions by 15% without data scaling achieved the best results.

There are four tables in this subsection, Table 4.2 and 4.3 show the results for data that was scaled before and after non-maximum suppression and Table 4.4 and 4.5 for data which was not scaled likewise. The reason for showing the results prior and after non-maximum suppression is to determine the non-maximum suppressions quality. Also, by showing both results prior and after shows that only by looking at the results after non-maximum suppression can we determine the best classification model. Each table consists of three different performance measures and the results are calculated for two standard deviations (95% confidence). Tables 4.3 and 4.5 contain an additional column which represents the average amount of difference in detections that is expected of the detector. The first number is calculated by averaging the absolute difference between number of ground truths

and number of detections. The numbers in the brackets indicate the difference represented by percentage. All images greatly vary in number of ground truths and a difference in percent is needed to realistically represent the differences between the ground truths and detections from our approach. The bold rows represent the best models obtained both by scaling and not scaling the data, independently in each table. The best models were chosen mainly by two criteria: by looking at the average difference in annotations and looking at the Precision and Recall values together. It is quite obvious from these results that better results were achieved by not scaling the data.

Experiment	Accuracy	Precision	Recall
Ordinary	0.67 ± 0.13	0.74 ± 0.22	0.69 ± 0.35
Ordinary normalized	$\mathbf{0.66 \pm 0.15}$	$\mathbf{0.67 \pm 0.27}$	$\mathbf{0.85 \pm 0.25}$
$\frac{x-\mu}{\sigma}$	0.67 ± 0.17	0.73 ± 0.21	0.68 ± 0.32
$\frac{x-\mu}{\sigma}$ normalized	0.60 ± 0.23	0.66 ± 0.25	0.68 ± 0.56
+5%	0.67 ± 0.13	0.75 ± 0.21	0.64 ± 0.27
+5% normalized	0.65 ± 0.19	0.71 ± 0.26	0.74 ± 0.41
+10%	0.68 ± 0.14	0.70 ± 0.23	0.66 ± 0.31
+10% normalized	0.64 ± 0.23	0.70 ± 0.25	0.56 ± 0.55
+15%	0.68 ± 0.11	0.69 ± 0.22	0.70 ± 0.24
+15% normalized	0.64 ± 0.22	0.68 ± 0.27	0.65 ± 0.54

Table 4.2: With data scaling (95% confidence)

Experiment	Accuracy	Precision	Recall	Difference
Ordinary	0.67 ± 0.12	0.67 ± 0.26	0.78 ± 0.30	312(25%)
Ordinary normalized	0.63 ± 0.22	0.60 ± 0.30	0.92 ± 0.16	445(28%)
$\frac{x-\mu}{\sigma}$	0.67 ± 0.12	0.66 ± 0.26	0.78 ± 0.28	288(23%)
$\frac{x-\mu}{\sigma}$ normalized	0.59 ± 0.20	0.59 ± 0.26	0.80 ± 0.50	472(31%)
+5%	0.69 ± 0.10	0.70 ± 0.26	0.75 ± 0.26	265(22%)
+5% normalized	0.64 ± 0.18	0.64 ± 0.30	0.84 ± 0.32	396(29%)
+10%	0.68 ± 0.12	0.66 ± 0.28	0.77 ± 0.26	275(24%)
+10% normalized	0.65 ± 0.20	0.66 ± 0.28	0.69 ± 0.52	378(33%)
+15%	$\mathbf{0.68 \pm 0.12}$	$\mathbf{0.66 \pm 0.26}$	$\mathbf{0.81 \pm 0.22}$	**249(21%)**
+15% normalized	0.64 ± 0.20	0.64 ± 0.30	0.78 ± 0.46	369(30%)

Table 4.3: With data scaling after non-maximum suppression (95% confidence)

Experiment	Accuracy	Precision	Recall
Ordinary	0.72 ± 0.10	0.69 ± 0.20	0.87 ± 0.13
Ordinary normalized	$\mathbf{0.72 \pm 0.11}$	$\mathbf{0.69 \pm 0.20}$	$\mathbf{0.88 \pm 0.13}$
$\frac{x-\mu}{\sigma}$	0.69 ± 0.08	0.74 ± 0.22	0.70 ± 0.11
$\frac{x-\mu}{\sigma}$ normalized	0.67 ± 0.07	0.71 ± 0.23	0.71 ± 0.12
+5%	0.71 ± 0.11	0.69 ± 0.20	0.88 ± 0.12
+5% normalized	0.72 ± 0.11	0.69 ± 0.20	0.88 ± 0.13
+10%	0.72 ± 0.08	0.69 ± 0.20	0.77 ± 0.19
+10% normalized	0.72 ± 0.08	0.68 ± 0.20	0.78 ± 0.18
+15%	0.72 ± 0.07	0.69 ± 0.20	0.77 ± 0.18
+15% normalized	0.72 ± 0.07	0.69 ± 0.20	0.78 ± 0.17

Table 4.4: Without data scaling (95% confidence)

Experiment	Accuracy	Precision	Recall	Difference
Ordinary	0.67 ± 0.14	0.61 ± 0.24	0.94 ± 0.10	355(25%)
Ordinary normalized	0.66 ± 0.16	0.61 ± 0.24	0.94 ± 0.08	380(26%)
$\frac{x-\mu}{\sigma}$	0.70 ± 0.10	0.67 ± 0.26	0.82 ± 0.10	249(20%)
$\frac{x-\mu}{\sigma}$ normalized	0.67 ± 0.12	0.63 ± 0.26	0.87 ± 0.10	308(23%)
+5%	0.67 ± 0.16	0.61 ± 0.24	0.95 ± 0.08	345(28%)
+5% normalized	0.67 ± 0.16	0.62 ± 0.24	0.94 ± 0.08	338(24%)
+10%	0.70 ± 0.12	0.64 ± 0.24	0.89 ± 0.14	205(17%)
+10% normalized	0.70 ± 0.12	0.63 ± 0.24	0.90 ± 0.12	211(17%)
+15%	$\mathbf{0.71 \pm 0.12}$	$\mathbf{0.65 \pm 0.24}$	$\mathbf{0.89 \pm 0.12}$	**194(16%)**
+15% normalized	0.70 ± 0.12	0.64 ± 0.24	0.90 ± 0.12	194(16%)

Table 4.5: Without data scaling after non-maximum suppression (95% confidence)

The best achieved detector was the one with +15% increased proposed regions size, without data scaling. To address another important question whether the results obtained from our approach are good or not, we have also tested an ACF model as a detector. The results are presented in Table 4.6. One very important note is that the ACF accuracy is incredibly high because it is classifying a tremendous amount of sliding windows and the true negatives dominate the confusion matrix. If we measured this differently by looking at the TP, FP and FN values at the confusion matrix and setting TN=0, we would get 0.51 ± 0.22 as a result. We can conclude from this table that our approach achieved far better results than an ACF detector alone. The criteria for this conclusion was the last column, the average difference in polyps and the Precision and Recall equally important. Accuracy is not a suitable measure for comparing these two approaches.

	Accuracy	Precision	Recall	Difference
ACF detector	0.99 ± 0.01	0.55 ± 0.27	0.90 ± 0.09	653(61%)
Our approach	$\mathbf{0.71 \pm 0.12}$	$\mathbf{0.65 \pm 0.24}$	$\mathbf{0.89 \pm 0.12}$	**194(16%)**

Table 4.6: Baseline comparison between two detectors on the dataset.

4.5.3 Qualitative evaluation

Besides having achieved very promising results in form of numbers, the detections needed to be further inspected. In order to do so, Image 05.01 was chosen because it is perhaps the clearest image of the whole dataset and the quality of results will be easily evident on it. Figures 4.12 - 4.17 are used to visualize the results in different ways. Figure 4.12 shows the output of the ACF regions proposal. These regions are further analyzed in Figure 4.13, which contains all region proposals which have $\tau > 0.2$ with a ground truth, plotted in red. The purpose for plotting Figure 4.13 is to show the level of correspondence between the red proposed regions and the ground truths in an image. It is visually noticeable that the proposed regions correspond well.

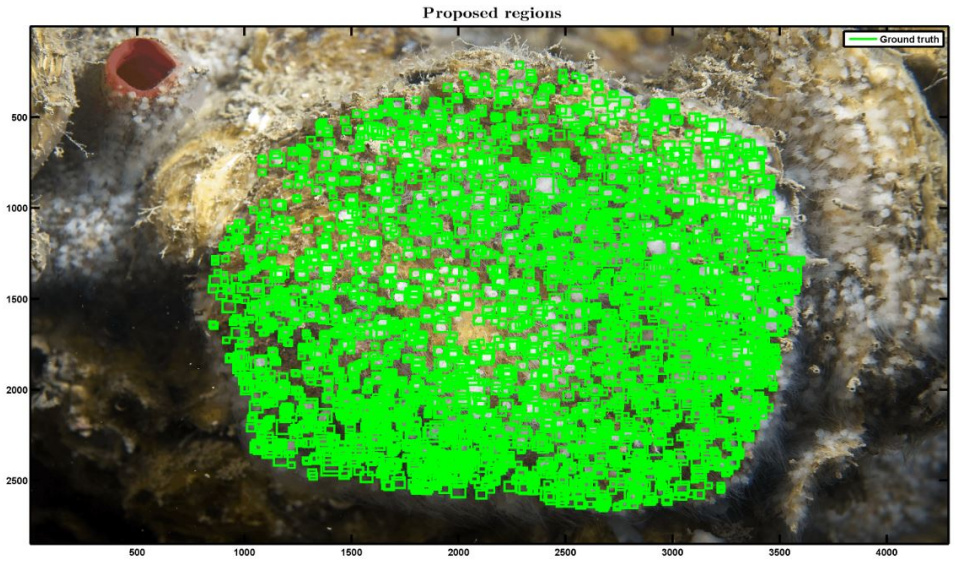


Figure 4.12: Visual results from the first part of the pipeline, the ACF. The green rectangles are its outputs after a non-maximum suppression with low threshold.

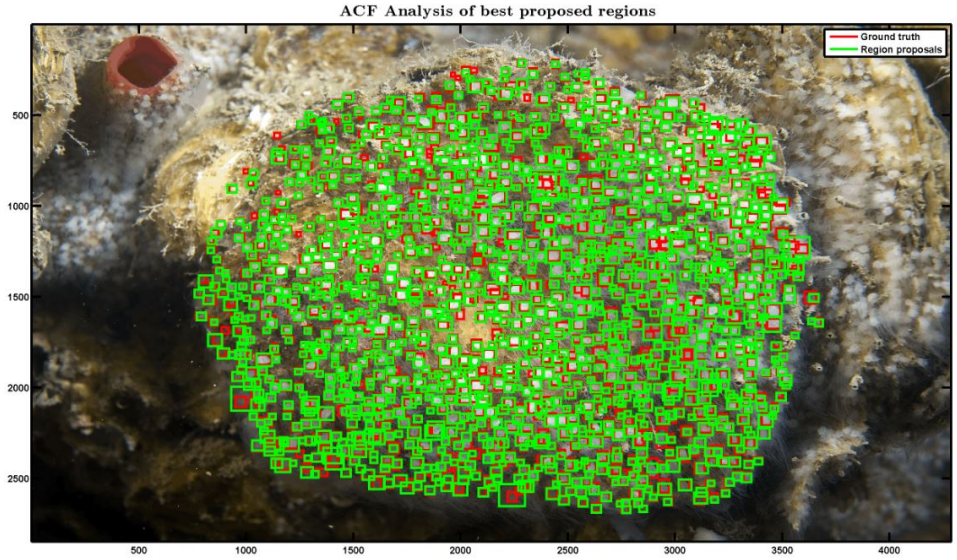


Figure 4.13: Visual results from ACF. The image has 1325 annotations and 4545 proposed regions of which 3030 lie within the polygon.

All annotations are plotted in red besides their best proposed region matches, which are colored green. The statistics for the image are quite interesting and show that for this particular image there are around 2.3 times more proposed regions then annotations in the polygon, 81% of the regions have $\tau > 0.2$ with average τ quality of 0.53. Finally, 95% of the annotations have a proposed region with $\tau > 0.2$ in this particular image.

Figures 4.14 and 4.15 show the detections within the polygon that were classified as positive and the ones classified as negative respectively. The three quality measures suggest that the model is strong. To renew the meaning of these quality measures and intuitively explain them, 89% of the all ground truths have been correctly classified, 81% of all positive examples have been accurately classified as positive and the detector has properly classified 76% of the regions. It is apparent from these figures that there are sometimes more than one regions per polyp and therefore a non-maximum suppression is indeed obligatory.

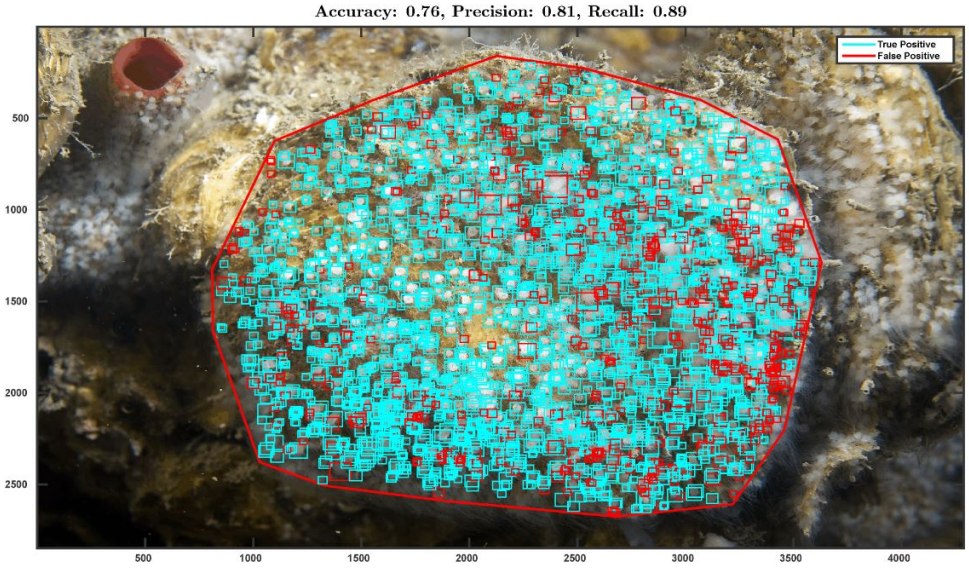


Figure 4.14: An illustration of the regions that the SVM classified as positive with three quality measures. The cyan rectangles are TPs, the red ones are FPs.

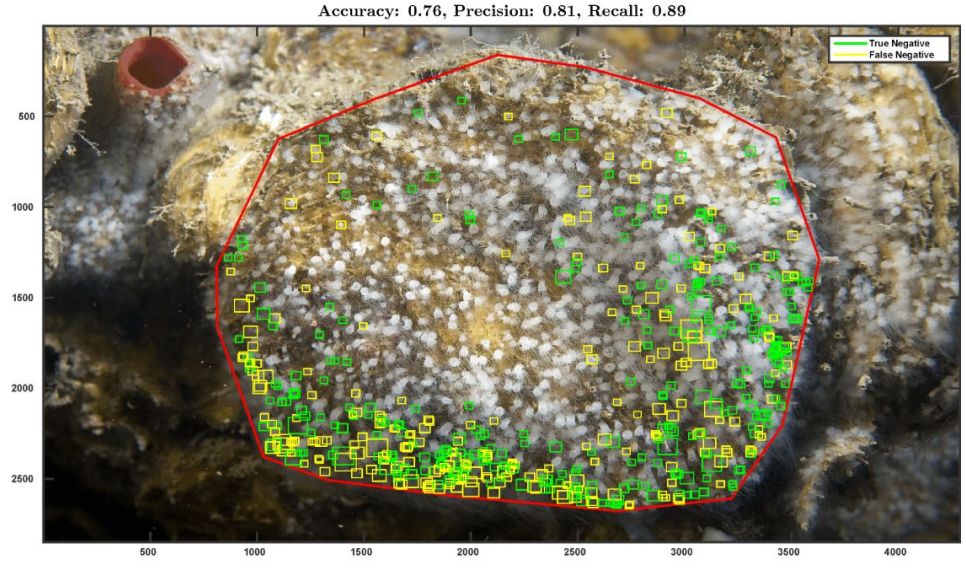


Figure 4.15: An illustration regions which SVM classified as negative along with three quality measures. The green are TNs, the yellow ones are FNs.

After that, Figures 4.16 and 4.17 illustrate the final detection results after a non-maximum suppression. The image used for representation of the results clearly has results among the best, and the quality of the image has a lot to do with it. After careful inspection, it is hardly noticable but among the FP detections there are rectangles that are actually part of a polyp, which means that the results of these pictures are even better numerically. At first glance the reader may think that it means that a slightly stricter non-maximum suppression threshold has to be set. Although, this is not the case because there are frequent occurrences of regions where the polyps are distributed incredibly densely. We can conclude from the visualizations of detections after a non-maximum suppression that the polyps along the edges of the oyster are detected with lower accuracy then the ones closer to the center, i.e., the cameras focus.

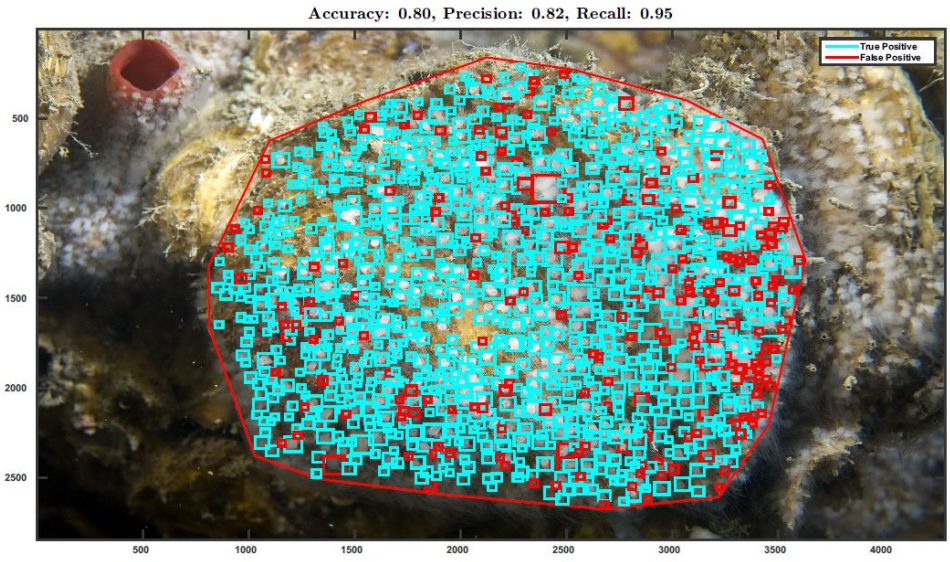


Figure 4.16: Visualization of the positive examples after a non-maximum suppression, as well as three quality measures.

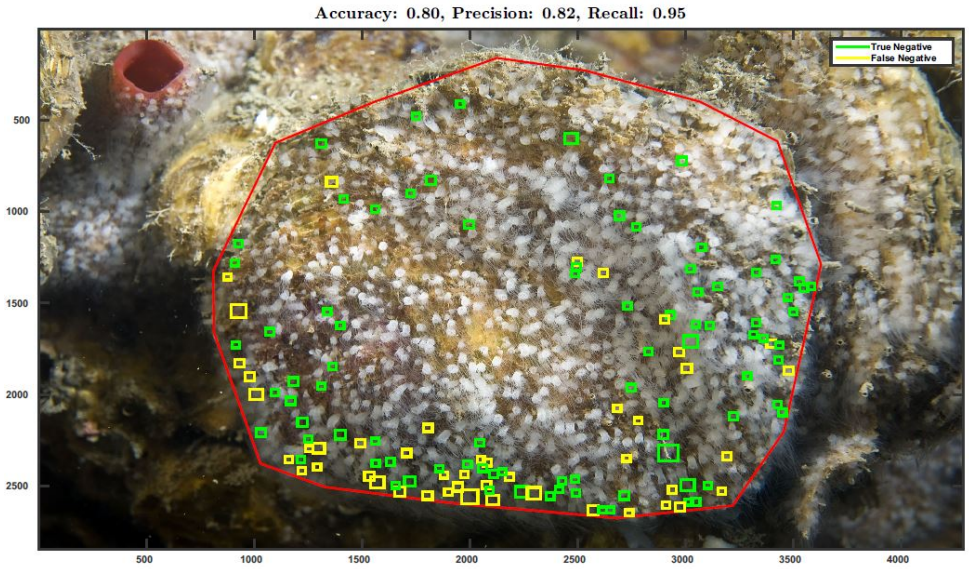


Figure 4.17: Visualization of the negative examples after a non-maximum suppression together with three quality measures.

After analyzing the quality of the proposed pipeline of algorithms, it is also important to show the time it used in order to achieve these results. The images are quite big and heavy with information as an input to an object detection solution, that is why our approach needed on average 2 minutes to detect the polyps in an image. ACF needed a little around 1 minute in order to output the regions, our CNN took nearly 1 minute as well and the SVM and the final non-maximum suppression took only a few seconds to finish the computing. However, the computational time was not as relevant as detecting these polyps as accurately as possible. The intention of this approach is not to achieve real-time detections, but rather accurate ones in a reasonable amount of time. To summarize, the model of greatest quality on average the number of output detections and the amount of ground truths differed by 16.3%.

4.5.4 Potential improvements

This subsection is intended to present one possible addition that would make this approach more powerful and practically useful. After obtaining the best model, we analyzed whether it was biased in any way. We were specifically looking at correspondence between the number of detections D outputted by our approach at images with different amounts of ground truths G

$$\rho = \frac{D}{G} - 1. \tag{4.1}$$

For example, if $\rho = 0.20$ for some image, it means that it has 20% more detections than the actual amount of polyps.
If the detector was extremely accurate, the red points in Figure 4.18 would lie on the $y = x$ line. However, the number of detections is always different from the number of ground truths. Accordingly, from Figure 4.19 we can conclude that there is a dynamic bias. The detector tends to output less detections when there are fewer ground truths, and to output more detections in images with a lot of ground truths. From these analysis we can conclude that the detector can be further improved using regression. A potentially improved version of our approach with a regression model would be able to approximate the number of polyps in an image more precisely.

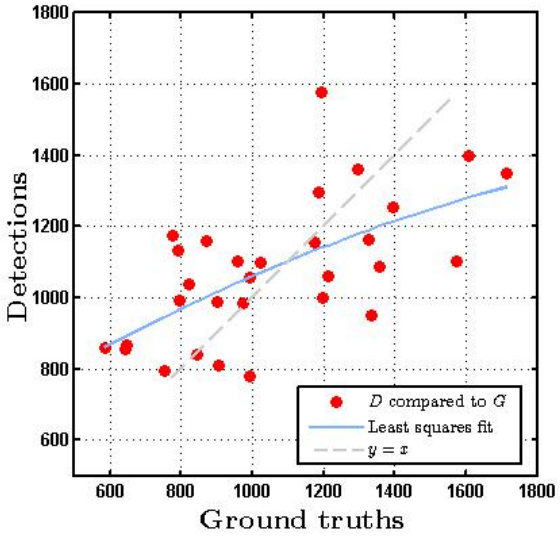


Figure 4.18: Illustration of the difference in number of ground truths and detections, in addition to a quadratic least squares fit to represent the trend.

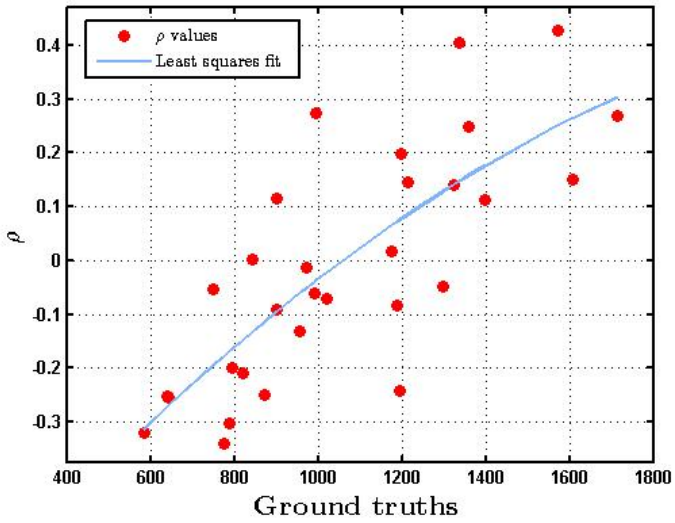


Figure 4.19: Illustration of the ρ bias values, in addition to a quadratic least squares fit to represent the trend.

Chapter 5

Conclusion

We have proposed a new approach for object detection in images. In this particular thesis, we aimed to accurately detect polyps in underwater images. The process began with the aggregated channel features (ACF) [16] algorithm which proposed regions that might contain polyps. Those regions had their features extracted by a CNN [28], which afterwards were labeled by a SVM classifier [12]. Finally, the regions were filtered once more by applying a non-maximum-suppression.

5.0.5 Strengths and weaknesses

Our approach was tested on a newly annotated one-of-a-kind dataset of underwater images of polyps. We have achieved excellent results despite the difficulties presented by the uncontrolled underwater environment. There are both pros and cons for the pipeline of algorithms that we used. The main weakness was the inability to accurately detect the polyps in the extremely dense and occluded areas. Partial occlusion of objects in images is a very difficult problem to tackle, not just for our detector, but generally speaking for all computer vision applications. One advantage of having an approach that was specifically trained for detecting polyps over a standard object detector like ACF is the detector quality and ability to further improve the results by many experiments for each algorithm we use. Once an algorithm converges to its maximum capability, that is it. On the other hand, dividing the whole process into several phases increases the margin for improvement.

We have shown that our approach achieves a lot better results then using an ACF trained detector (not as regions proposal) for the task. Also, our approach has a big advantage over methods that just estimate the number of objects in the image and not localizing them, because visualizing the detections makes the debugging and improvement a lot simpler and the outputs are easier to interpret. It also allows the possibility for a human operator to manually correct the detections and produce close to exact numbers.

5.0.6 Future work

Our approach can be greatly improved by training a CNN specifically on our dataset and also support our approach with a regression method.

Moreover, the classifier can be further improved by training it with augmented dataset by translating and rotating the annotated polyps. In addition, future work may also include pre-processing the proposed regions before extracting their features.

This thesis produced a general object detection approach that is applicable to any object detection problem. It would be therefore compelling to see this approach tested on many different multi-label object detection domains and hopefully inspire further constructive research in the vast domain of computer vision.

Bibliography

[1] Bogdan Alexe, Thomas Deselaers, and Vittorio Ferrari. Measuring the objectness of image windows. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(11):2189–2202, 2012.

[2] Naomi S Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[3] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Computer vision–ECCV 2006*, pages 404–417. Springer, 2006.

[4] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[5] Antoni B Chan, Zhang-Sheng John Liang, and Nuno Vasconcelos. Privacy preserving crowd monitoring: Counting people without people models or tracking. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–7. IEEE, 2008.

[6] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[7] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531*, 2014.

[8] Dan Cireşan, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber. A committee of neural networks for traffic sign classification. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 1918–1921. IEEE, 2011.

[9] Dan Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012.

[10] Dan C Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, pages 1237–1242, 2011.

[11] Dan Claudiu Cireşan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. Convolutional neural network committees for handwritten character classification. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 1135–1139. IEEE, 2011.

[12] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[13] James L. Crowley and Olivier Riff. Fast computation of scale normalised gaussian receptive fields. In *Proceedings of the 4th International Conference on Scale Space Methods in Computer Vision*, Scale Space'03, pages 584–598, Berlin, Heidelberg, 2003. Springer-Verlag.

[14] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[15] Piotr Dollár. Piotr's Computer Vision Matlab Toolbox (PMT). `http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html`.

[16] Piotr Dollár, Ron Appel, Serge Belongie, and Pietro Perona. Fast feature pyramids for object detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36(8):1532–1545, 2014.

[17] Piotr Dollar, Serge Belongie, and Pietro Perona. The fastest pedestrian detector in the west. In *Proceedings of the British Machine Vision Conference*, pages 68.1–68.11. BMVA Press, 2010. doi:10.5244/C.24.68.

[18] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[19] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.

[20] Elena A Fedorovskaya, Frans JJ Blommaert, and Huib de Ridder. Perceptual quality of color images of natural scenes transformed in cieluv color space. In *Color and Imaging Conference*, volume 1993, pages 37–40. Society for Imaging Science and Technology, 1993.

[21] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *ICML*, volume 96, pages 148–156, 1996.

[22] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jagannath Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 580–587. IEEE, 2014.

[23] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.

[24] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1998.

[25] Sara Hočevar. *Populacijska dinamika polipov uhatega klobučnjaka (Aurelia aurita) v koprskem zalivu: zaključna naloga.* PhD thesis, S. Hočevar, 2013.

[26] Sara Hočevar, Tjaša Kogovšek, and Malej Alenka. Interannual variation of the sexual reproduction of Aurelia aurita followed in situ (Bay of Koper, Adriatic sea). *Abstracts volume : 49th European Marine Biology Symposium*, 2014.

[27] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification, 2003.

[28] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[29] Ian Jolliffe. *Principal component analysis.* Wiley Online Library, 2002.

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[31] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face recognition: A convolutional neural-network approach. *Neural Networks, IEEE Transactions on*, 8(1):98–113, 1997.

[32] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10), 1995.

[33] Victor Lempitsky and Andrew Zisserman. Learning to count objects in images. In *Advances in Neural Information Processing Systems*, pages 1324–1332, 2010.

[34] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[35] Alenka Malej, Tjaša Kogovšek, Andreja Ramšak, and Luca Catenacci. Blooms and population dynamics of moon jellyfish in the northern Adriatic. *Cahiers de biologie marine*, 53(3):337–342, 2012.

[36] Masakazu Matsugu, Katsuhiko Mori, Yusuke Mitari, and Yuji Kaneda. Subject independent facial expression recognition with robust face detection using a convolutional neural network. *Neural Networks*, 16(5):555–559, 2003.

[37] Yu. Nesterov. Gradient methods for minimizing composite functions. *Mathematical Programming*, 140(1):125–161, 2013.

[38] Petter Ögren, Edward Fiorelli, and Naomi Ehrich Leonard. Cooperative control of mobile sensor networks: Adaptive gradient climbing in a distributed environment. *Automatic Control, IEEE Transactions on*, 49(8):1292–1302, 2004.

[39] John Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

[40] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[41] Mikel Rodriguez, Ivan Laptev, Josef Sivic, and Jean-Yves Audibert. Density-aware person detection and tracking in crowds. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2423–2430. IEEE, 2011.

[42] Stuart Russell and Peter Norvig. Artificial intelligence: a modern approach. 1995.

[43] Robert E Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

[44] Bernhard Scholkopft and Klaus-Robert Mullert. Fisher discriminant analysis with kernels. *Neural networks for signal processing IX*, 1:41–48, 1999.

[45] Patrick Sudowe and Bastian Leibe. Efficient use of geometric constraints for sliding-window object detection in video. In *Computer Vision Systems*, pages 11–20. Springer, 2011.

[46] Marko Tkalcic, Jurij F Tasic, et al. Colour spaces: perceptual, historical and applicational background. In *Eurocon*, pages 304–308, 2003.

[47] Jasper RR Uijlings, Koen EA van de Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[48] Koen EA Van de Sande, Jasper RR Uijlings, Theo Gevers, and Arnold WM Smeulders. Segmentation as selective search for object recognition. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 1879–1886. IEEE, 2011.

[49] Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

[50] Zhenhua Wang, Bin Fan, and Fuchao Wu. Local intensity order pattern for feature description. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 603–610. IEEE, 2011.

[51] Wikipedia. Cieluv — wikipedia, the free encyclopedia, 2015. [Online; accessed 27-August-2015].

[52] Wikipedia. Gradient — wikipedia, the free encyclopedia, 2015. [Online; accessed 2-September-2015].

[53] Bin Yang, Junjie Yan, Zhen Lei, and Stan Z Li. Aggregate channel features for multi-view face detection. In *Biometrics (IJCB), 2014 IEEE International Joint Conference on*, pages 1–8. IEEE, 2014.

[54] C. Lawrence Zitnick and Piotr Dollár. Edge boxes: Locating object proposals from edges. In *ECCV*, pages 391–405, 2014.